Techniques and Countermeasures of TCP/IP OS Fingerprinting on Linux Systems

by

Riaan Stopforth

Submitted in fulfilment of the academic
requirements for the degree of
Master of Science in the
School of Computer Science,
University of KwaZulu – Natal,
Durban

December 2007

Abstract

Port scanning is the first activity an attacker pursues when attempting to compromise a target system on a network. The aim is to gather information that will result in identifying one or more vulnerabilities in that system. For example, network ports that are open can reveal which applications and services are running on the system. How a port responds when probed with data can reveal which protocol the port utilises and can also reveal which implementation of that protocol is being employed. One of the most valuable pieces of information to be gained via scanning and probing techniques is the operating system that is installed on the target. This technique is called operating system fingerprinting.

The purpose of this research is to alert computer users of the dangers of port scanning, probing, and operating system fingerprinting by exposing these techniques and advising the users on which preventative countermeasures to take against them.

Analysis is performed on the Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Protocol (IPv4 and IPv6), and the Internet Control Message Protocol (ICMPv4 and ICMPv6).

All the software used in this project is free and open source. The operating system used for testing is Linux (2.4 and 2.6 kernels). Scanning, probing, and detection techniques are investigated in the context of the Network Mapper and Xprobe2 tools.

Preface

The experimental work described in this dissertation was carried out in the School of Computer Science, University of KwaZulu-Natal, Durban, from January 2006 to March 2007, under the supervision of Mr Luke Vorster and Dr David Erwin.

These studies represent original work by the author and have not otherwise been submitted in any form for any degree or diploma to any tertiary institution. Where use has been made of work of others it is duly acknowledged in the text.

The content of this thesis has been summarised in a paper called "*Counter-measures for operating system fingerprinting*" (Stopforth *et. al.*). This paper was accepted by the *EngineerIT – Electronics, computer, information & communications technology in engineering* journal, and was published in the April 2007 publication.

Table of Contents

Acknowledgements

A lot of help and support has been received from a number of people.

I would like to express my thanks to the following people:

- **Evgeniy Polyakov** for the help he provided while the tests were performed with the OSF module. It is amazing to find somebody that is willing to spend a large amount of their time to assist a person with their research.

- My supervisors, **Mr Luke Vorster** and **Dr David Erwin**, as well as all the members at the University of KwaZulu Natal, who have guided and assisted me during the course of my research.

- **My family**, who have supported me during the research period, especially to my mother who has proofread my documentations, even though at times it must have been difficult trying to understand what I was attempting to convey.

- Finally, to **Jesus Christ** my saviour, who has helped me in all the research, problems, achievements and results. Without him, I will not have been able to complete any of my life tasks.

Overview

A synopsis of the events that took place in this project are:

1. A broad investigation into security notions pertaining to common computer security threats. For example, email spamming, password guessing, computer viruses, computer intrusion, and social engineering. The investigation covered well-known techniques and related countermeasures.

2. Gained awareness and deeper understanding of data communications and networking protocols, architecture, and implementations.

3. Acquired technical orientation within the Linux operating system in terms of architecture, system commands, administration, and security features. This was solidified by completing the Linux Professional Institution (LPI) exams and acquiring the related certification.

4. Conducted a deeper investigation of the techniques and related countermeasures of computer hacking, with particular focus on Linux and inter-networking vulnerabilities.

5. Investigated hacking methodologies with the intent of identifying a specific phase that is of interest. Operating system fingerprinting was identified as a crucial activity that would be of much value to counteract – it is currently a nascent field.

6. Investigated the various tools and techniques currently employed during the phase of scanning, probing, and operating system

fingerprinting. Determined what information an attacker receives from a target via these methods.

7. Investigated and experimented with techniques that could be used to modify the Linux operating system in order to prevent it from being fingerprinted on a network.

8. Investigated and experimented with the Linux firewall (iptables), its configuration, and its modules that are currently available to aid preventing or deterring fingerprinting techniques.

An outline of the content of the dissertation is as follows:

1. Should the reader have little knowledge on the subject, the dissertation begins with an introduction to the domain of the project. This includes a background to data communications, information security, and a discussion that highlights the significance of this research.

2. An overview of computer network architectures and relevant networking protocols is provided as this knowledge is required to understand the hacking techniques relevant to this project.

3. The techniques applied by operating system fingerprinting tools, particularly the popular Nmap and Xprobe2 applications, is provided.

4. The results of tests and experiments that were performed with the tools is presented.

5. The characteristics that make it possible for scanning and probing tools to fingerprint the Linux operating system is discussed.

6. A preliminary set of countermeasures is proposed. These countermeasures primarily involve the modification of characteristics, identified in the previous section, of the TCP/IP stack. The results of investigating these countermeasures is also presented.

7. Various techniques employed by operating system fingerprinting tools can be detected by the target system, thus providing an opportunity for reactive countermeasures. These techniques are discussed and the results of experimentation performed when these techniques are applied is also presented.

8. The dissertation concludes with a final list of valid countermeasures, presented in order of significance, and a brief outline of further work that should be undertaken as a result of the results of this project.

Introduction

The primary reason for a communication system is the exchange of information between two parties. Communication systems are so deeply ingrained into modern life that we almost take them for granted. As a result of an ongoing need for more efficient and capable communication systems, the fields of computer science and data communications merged during the 1970s and 1980s. Networking is a resulting field that concerns itself with the technology and architecture of the communication systems used to interconnect communicating devices. The emergence of this field had a radical impact on technology, products, and companies of the computer industry. [Stallings 2004a]

Most computer networks are, by nature, shared resources used by many applications for many different purposes. Sometimes the data transmitted between applications is confidential. In these situations, security measures are applied to restrict access to the information while it is being exchanged via the data transmission system. Access restrictions are also applied to the information while it is being stored on the computers themselves. [Peterson 2003] One of the more undesirable side-effects of using shared computer networks, such as the Internet, for the exchange of confidential information is the ever growing security threat that confidential information can be stolen.

By the nature of the fact that more and more computing resources can be utilised remotely, there is another equally significant and increasing security threat – computing resources can be hijacked, or vandalised.

Security experts accept that it is an ongoing battle to secure an information system – an information system has to provide at least some access to itself, and the information on it, in order to be of any use. This access can be obtained by garnering security information, such as access passwords, from valid users of the system. Passwords can be bought, stolen, or managed carelessly. The system could also have flaws, or bugs, that make it vulnerable to other forms of attack. Security experts work in a grey area, only ever able to answer the question: is it secure enough? [Schneier 2000]

Information security is a volatile field, particularly in the context of data communication networks. Computer technology progresses exponentially over time, according to Moore's Law, and techniques for exploiting these technologies are realised at an equally alarming rate.

Some examples of security exploits, obtained from a news survey [Schneier 2000], that occurred over a period of just one month – March 2000 – are:

- The SalesGate.com website was broken into. Almost 3000 customer records, including credit card numbers and personal information, was stolen.

- Convicted criminal hacker Kevin Mitnick testified before Congress. He told them that social engineering is a major security vulnerability. He said he can often obtain passwords and other secrets just by pretending to be someone else and asking for the information.

- Pierre-Guy Lavoie, 22, was convicted in Quebec of breaking into several Canadian and US government computers. He served 12 months in prison.

- Japan's Defense Agency delayed the development of a new defense computer system after it was discovered that the software had been developed by members of the Aum Shinrikyo cult.

- An email virus, or worm, called Pretty Park, spread across the

Internet. It was not the first of its kind – it spreads automatically by sending itself to all the addresses listed in the user's Outlook Express address book.

- Guiseppe Russo and his wife, Sandra Elazar, were arrested in Sicily after stealing about 1000 credit cards on the Internet.

- A hacker, a.k.a. 'Coolio', denied launching massive denial-of-service attacks in February 2000. he admitted to hacking about 100 websites in the past, including cryptography company RSA Security and a website belonging to the US State Department.

- Attackers launched a denial-of-service attack on Microsoft's Israeli website.

- Jonathan Bosnak, a.k.a. 'The Gatsby' was sentenced to eighteen months in prison for hacking into three separate telephone company systems.

- The military of Taiwan announced that it had discovered more than 7000 attempts by Chinese hackers to enter the country's security systems.

Some examples of software vulnerabilities discovered [Schneier 2000] during that same time period of March 2000 are:

- A vulnerability was discovered in the Microsoft Internet Explorer 5.0 that allows an attacker to set up a web page giving her the ability to execute any program on a visitor's machine.

- By modifying a URL, an attacker can completely bypass the authentication mechanisms protecting the remote management screens of the Axis StarPoint CD-ROm servers.

- If an attacker sends a Netscape Enterprise Server 3.6 a certain type of long message, a buffer overflow crashes a particular process. The attacker can then execute arbitrary code remotely on the server.

- It is possible to launch a denial-of service attack that Internet Security System's RealSecure Network Intrusion Detection software fails to detect.

- By sending a certain URL to a server running Allaire's ColdFusion product, an attacker can receive an error message giving information about the physical paths to various files.

- Omniback is a Hewlett-Packard product that performs system backup routines. An attacker can manipulate the product to cause a denial-of-service attack.

- By manipulating the contents of certain variables, an attacker can exploit a vulnerability in DNSTools 1.0.8 to execute arbitrary code.

- If you send a long login name and password, even an incorrect one, to BisonWare's FTP Server 3.5, it will crash.

Any two points on the Internet are adjacent, regardless of geographical location. Attackers can choose a country with weaker computer crime laws to base their operations. Internet attackers don't have to be anywhere their target. This has enormous security implications and, therefore, was chosen as the general area to investigate in this research project. A security officer of a computer on the Internet has to worry about all the computer criminals in the world. The global nature of the Internet also complicates criminal investigation and prosecution. It is often unclear how to go about prosecuting a computer criminal in many cases – due to a lack of political borders on the Internet, it is unclear who should police it.

Hacking remote computer systems is considered illegal in most countries. In some countries, such as the U.S.A. and Singapore, the use of hacking techniques are considered analogous to using munition. Therefore, the techniques like those investigated in this project are closely guarded by those who use them. Hackers generally do not want their identities known, nor their hacking exploits publicised. Some, however, have been caught, convicted, and exposed to the public in the process. Others have changed their profession. For example, Dr. K is a hacker that does not want his real name known. He started the Phreak/Hack – United Kingdom (P/H-UK) e-zine and now works as a computer networking and security specialist. There are some “purist” hackers, that release software tools that implement their techniques for others to study. There are also some hackers that seek acknowledgement from their peers by releasing automated versions of their techniques in the form of software tools called 'exploits'. These tools are usually proof of the techniques they employ, but, on the darker side, they are often intended to be used by others. Hackers tend to proliferate their techniques by sharing with each other.

There is, therefore, not much pertinent academic literature available on the specific topic of research, apart from documents written by the authors of specific tools and the odd turned-professional such as Dr K. Regarding the public arena, many security professionals and researchers who do publish their work merely repeat the above-mentioned documents when covering the topic of this research project.

Port scanning is one such set of techniques. Port scanning, or probing, techniques are used to determine which services are running on a remote system and, more importantly, whether the services are in the listening state or not. These listening services, or open ports, are the entry points for networked applications, such as web browsers and email clients, to make use of the services on a remote system.

Port scanning techniques are used by system administrators to test and troubleshoot their networked systems, so there are many benign tools available for this purpose. Port scanning is a double-edged sword, however, because similar techniques are also used by hackers to gain information about systems they have become aware of and want to target. Open ports are, therefore, also the entry points attackers use to penetrate vulnerable systems.

For example, if an attacker knows which services, and which version of those services, are running on a target system, then an attack strategy to penetrate the target can be formulated. This is because software vendors and engineers publish the vulnerabilities of specific versions of services they produce or use. A mailing list that addresses issues experienced in using a particular vendor's product is a common place to begin this type of research. An attacker needs no more than one exploitable service to compromise a system.

For an attacker, there is far more valuable information to be gained by port scanning than simply determining which version of which services are running on a target. A subset of port scanning techniques, known as operating system fingerprinting, involves determining the type and version of the operating system employed by a target. This information is valuable for an attacker because operating systems have vulnerabilities too, and it is, in fact, the operating system that will ultimately be compromised if an attack is successful. In this regard, services are merely gateways to the operating system, and many successful attacks have been achieved through exploiting a combination of a service vulnerability, to gain remote entry to a system, and an operating system vulnerability.

An example of a hacker that used port scanning techniques in successful attacks is Kevin Mitnick. He was convicted for theft of documents and manuals from PacBell (1982), breaking into Digital Equipment Company's network (1998) and stealing credit card numbers from Netcom (1995) (Dr K 2000).

Examples of known attacks that can be performed once the open ports of a target system is known, are: Project Loki (daemon9 1996), ping flooding and IP spoofing, to name a few. With regards to operating system fingerprinting, examples of known attacks that can be performed on a fingerprinted operating system are described by McClure *et. al.* (2005)

It is vital for system administrators and individuals to countermeasure port scanning techniques as this could deter attackers from subsequent stages of their attack strategies or, even better, prevent attackers from formulating strategies altogether (due to a lack of sensitive information about target systems). Failure to provide scanning countermeasures could result in considerable loss. For example, the infiltration of important and confidential information, the corruption of data, and, in the worst case, loss of control of mission-critical systems such as medical or military systems.

Recall that system administrators use port scanning tools and techniques for testing and troubleshooting their networks. Though malignant port scanning must be counteracted, and ideally prevented, system administrators might find a secondary value in the techniques presented in this project – seeing their systems from the perspective of an attacker is the best way to determine the vulnerabilities and, therefore, the countermeasures they need to apply to secure their systems as much as possible. Once suitable countermeasures have been applied, the same scanning, probing, and fingerprinting techniques can then be used to validate the effectiveness of those countermeasures.

For the reasons mentioned above, this research project is justified as a valuable contribution to the ongoing endeavour of keeping up with the vulnerabilities and resultant dangers we expose ourselves to as we advance our networked computing technology.

Chapter 1: Protocols and Techniques

For information exchange between two systems to occur, they must be interconnected directly or via a communications network. A communications model is typically comprised of the following elements [Stallings 2004a]:

- **Source** – The device that generates data to be transmitted.
- **Transmitter** – The device that transforms and encodes the data so as to produce a signal that can be transmitted across some sort of transmission system.
- **Transmission System** – A single transmission line or a more complex network of transmission lines that ultimately connect the source and destination.
- **Receiver** – Accepts the signal from the transmission system and decodes it into a format that can be processed by the destination device.
- **Destination** – Receives the transmitted data.

Apart from the data to be transmitted, some additional data is required to allow for the communication to take place. The source, transmission system, and destination devices must perform other tasks [Stallings 2004b]:

- The source system must notify the communication network of the identity of the destination system.

- The source system must be sure that the destination system is prepared to receive data.

As a result, special information regarding the data must be added in a specific order with the data sent. *An agreement that specifies the format and meaning of messages computers exchange is known as a communication protocol* [Comer 2004].

Port scanning, probing, and fingerprinting involves the analysis of information about remote systems. This information is in the format specified by network protocols used on computer networks. The next section introduces network architecture and the various protocols that are significant in the context of this project. Scanning, probing, and fingerprinting techniques are then introduced.

1.1. Protocol Architecture

A layered protocol architecture is needed to allow for the development of standardised network technologies. [Stallings 2004a] The advantage of the layered approach is that the protocols of each layer are easy to manage and maintain in isolation. The lower layers can be changed without affecting the upper layers. For example, the layers that comprise internetworking technology, the architecture of the Internet, can be deployed on a wide variety of communication systems. Similarly, the upper layers can also reuse the functionality provided by the lower layers. For example, for most communication systems, three are a number of protocols that can be transmitted through them.

The OSI model and TCP/IP stacks are the two main layered architectures that are investigated in this project. OSI is an architecture model, while the TCP/IP stack is an implementation that happens to have a lot in common with the OSI model, even though they were developed independently. TCP/IP is the network architecture that enables the Internet.

A comparison of the two architectures and their corresponding layers is depicted in Figure 1.

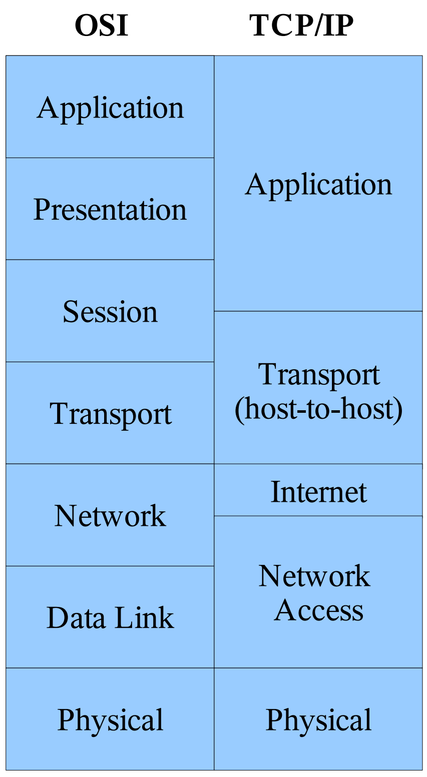


Figure 1 – Comparison of the OSI and TCP/IP Architectures

The layers that are focussed on in this project are the Transport, Internet and Network Access layers of TCP/IP stack.

1.1.1 Transmission Control Protocol (TCP)

Most of the information in this section has been taken from RFC 793 (Postel 1981a), which deals with the TCP protocol.

The TCP protocol corresponds with the Transport layer in Figure 1.

Applications that utilise the TCP protocol interact with the TCP stack. The TCP stack is a software module that interacts with the operating system. The operating system, in turn, interacts with the software driver of the network interface card (NIC). The NIC is directly connected to a local area network (LAN).

TCP is a connection-oriented end-to-end protocol. It is designed to be as reliable as possible, and achieves this reliability by providing transmission flow-control functionality. When a packet is transmitted a copy of the packet is put into the re-transmission queue and a timer is started. If an acknowledgement is not received after a specified time, then the packet is re-transmitted. It should be noted, though, that even if an acknowledgement is received within the time-window (the TCP layer has done its part), the packet has not necessarily been delivered to the other end of the connection. The format of the TCP packet header is presented in Appendix A.1. The TCP window is an indication to the sender as to the amount of data it can send to the receiver before the receiver's buffer overflows. As a result of the TCP stack reporting the window value, flow-control can be achieved by the sender. The window notifies the sender as to how many octets, starting with the acknowledgement number of that packet being sent, the receiver is willing to receive.

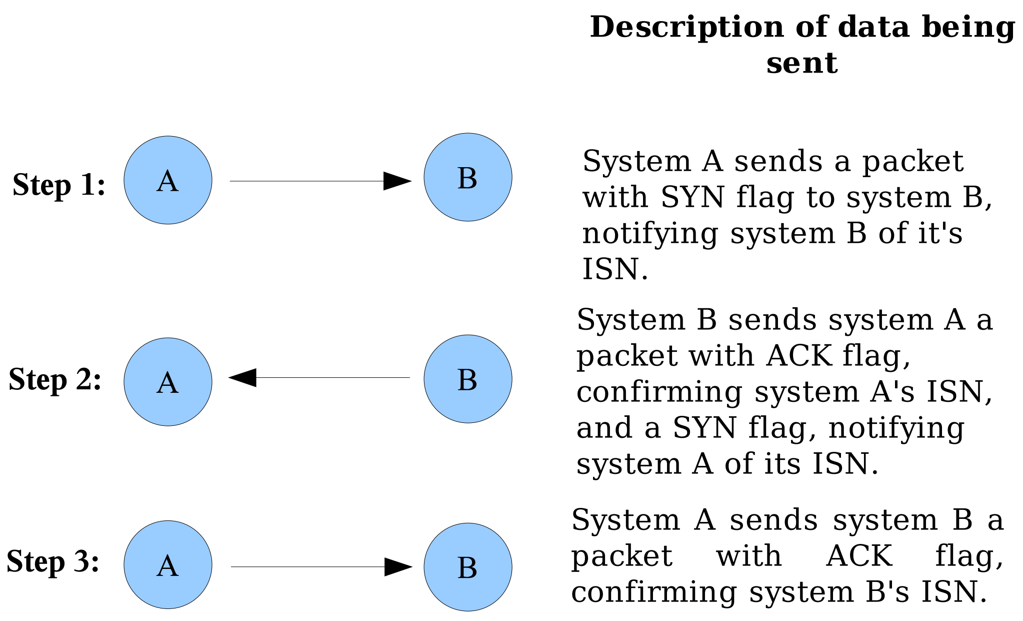
A TCP connection is a two-way connection – both hosts must establish their own end. This is achieved with the exchange of three messages, known as the three-way handshake, as depicted in Figure 2. The system that wants to initialise the connection (A), transmits a TCP packet with the synchronise (SYN) control flag set to 1. This is the first message. The receiving system (B) acknowledges the reception of the SYN packet from A and requests to initialise its own end of the connection. Therefore, the packet that is sent by B in response to A has both the SYN and acknowledge (ACK) control flags set to 1. This is the second message. A's end of the connection is established upon receipt of the second message. Finally, A sends a packet to B with the ACK control flag set to 1. This is the third message. B's end of the connection is now also established. To close a connection, a similar protocol is employed, only the FIN control flag is used instead of SYN.

1.1.1.1 Sequence Number Field

The purpose of the sequence number is to confirm, in an acknowledgement, that all packets from the most recently confirmed packets up to, but not including the most recently sent packet, have been successfully received. This is done via an acknowledge packet. Once an acknowledgement of a packet has been received, the copy in the re-transmission queue can be removed. The sequence number also identifies the order that incoming packets should be processed. This field cycles from 0 to 2^{32}-1 and therefore all arithmetic is done modulo 2^{32}.

The sequence number of the sender differs from that of the receiver. Therefore, the sender needs to notify the receiver of its initial send sequence number (ISN) and the receiver needs to notify the sender of its ISN. The initial receive sequence (IRS) number is the ISN sent by the receiving computer (system B in Figure 2). Once the sequence number of the other host has been received, the sequence number of that host must be confirmed. This process is explained in Figure 2, where A and B are considered to be separate systems.

The three-way handshake is needed as hosts are not synchronised and might have different ways of generating the ISNs.



Description of data being sent

System A sends a packet with SYN flag to system B, notifying system B of it's ISN.

System B sends system A a packet with ACK flag, confirming system A's ISN, and a SYN flag, notifying system A of its ISN.

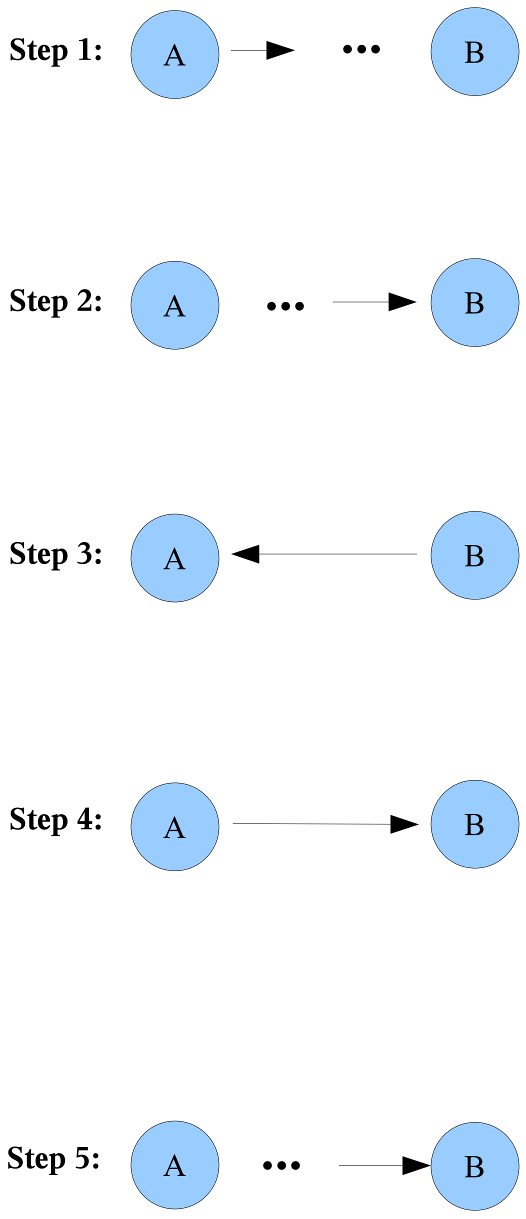
System A sends system B a packet with ACK flag, confirming system B's ISN.

Figure 2 – A description of the "Three-way handshake"

1.1.1.2 Opening and Closing Connections

The establishment of a TCP connection is based on the three-way handshake, as shown in Figure 2. Figure 3 elaborates on this, considering a situation where system B received an old packet that was still floating around in the network, after system A crashed. The initial state of system A is closed and system B is listening. After step 5 in Figure 3, the sequence of steps 2 to 3 in Figure 2 will occur.

This situation does not usually occur, as an engineering estimation has been made that a system should be connected to a network after it has crashed after a delay time of 2 minutes (Postel 1981a). The time that a packet lives is considered to be less than this delay value.



Description of data being sent

System A sends a SYN packet to system B with a sequence number with a value of 100.

System B receives a SYN packet from system A that has a sequence number with a value of 90.

System B replies to system A with a packet that has a sequence number of 300, and acknowledges that it is expecting the next packet from system A to be one with sequence number 91.

System A recognizes that there must be an error, as system B is not awaiting the same sequence number that system A is willing to send, and therefore a RST packet is sent to system B, putting it back into a listening mode.

System B eventually receives the original SYN packet from system A that has a sequence number with a value of 100.

Figure 3 – The initial stages when an old packet is received

A similar situation occurs in the event of a system that crashes. A crashed system is one that does not respond due to an abnormal event that has occurred. In this scenario, the crashed system or the related daemon on the system might have to be restarted before subsequent connections can be established.

If system A crashes (assuming that the daemon of the TCP control in system A had to restart), then system B is still under the impression that it has an established connection to system A. System A then sends a SYN packet to system B, in which instance system B will ignore it, as it is already connected to system A. System B continues to send data from the prior connection, but system A will realise something is wrong as it is expecting a SYN/ACK packet acknowledging the sequence number it has sent recently. System A responds to system B with an RST packet, which closes the connection with system B, and a new connection is established. This is known as a half-open connection.

Other scenarios can occur other than the example shown above. According to RFC 793 (Postel 1981a) all scenarios have to follow the rules as to when an RST packet is generated. These rules are not always obeyed as some operating systems don't follow them. RST packets are sent in the following conditions:

- If the connection does not exist, and packets are received from a system.

- Should the ports that the receiving SYN packet is trying to connect to be closed.

- The receipt of an ACK packet without establishing the connection. As seen later in the document, these rules are not always obeyed as some operating systems don't follow them for either security or ignorance reasons.

- The receipt of an ACK packet that has a different sequence number than that expected.

- If there is an unacceptable level in security, precedence or compartment. This could involve a user trying to login to a system, but they do not have rights to that system, or an incorrect password has been entered.

If an RST packet is sent after a sequence number has been initiated, then the sequence number of the RST packet is taken from the acknowledgement field of the incoming packet, otherwise the sequence number is taken to be zero.

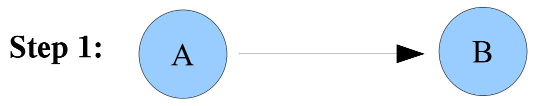
The RST packets are validated by checking the sequence number of the packets. Once the RST packet is confirmed to be valid and the system is in the listen state, it ignores it. Should it be in a SYN-received state (from a prior listen state) the system should return to a listen state, otherwise the system changes to a closed state.

A connection is closed when no further data is needed to be transported between two systems. On closing a connection, an action similar to the three-way handshake is implied.

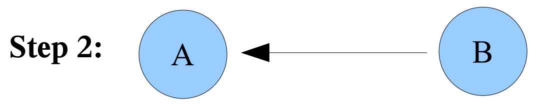
This is illustrated in Figure 4. System A and system B have a connection established between them. System A is currently on a sequence number with a value of 99 and system B is currently on a sequence number with a value of 299. In Figure 4, step 2 and step 3 is often combined into a single step.

The TCP protocol and previous examples explain the basic communication process that occurs in a network.

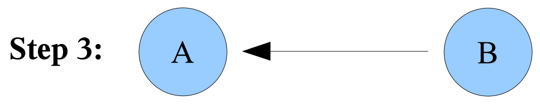
Description of data being sent



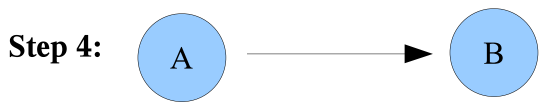
System A wants to close the connection, and goes into the FIN-WAIT-1 state. A FIN/ACK packet is sent from system A to system B that has a sequence number of 100 and acknowledges that it is waiting for a packet with a sequence number with a value of 300.



System B goes into the CLOSE-WAIT state on reception of FIN packet, and replies with an ACK packet (that has a sequence number with a value of 300) that acknowledges that it is waiting for the reception of packet 101. System A goes into FIN-WAIT-2 state on receipt of the packet.



System B is ready to close and goes into a LAST-ACK state and therefore sends a FIN/ACK packet with a sequence number to system A with a value of 300, and acknowledges that it is waiting to receive a packet with a sequence number with a value of 101.



System A goes into the TIME-WAIT state and sends an ACK packet to system B with a sequence number that has a value of 101, and acknowledges that it is ready to receive a packet with a sequence number with a value of 301. System B closes on reception of this packet, in which system A then closes after twice the time of the maximum packet lifetime.

Figure 4 – Closing "three-way handshake"

TCP is one of the commonly used protocols in port scanning and probing techniques.

1.1.2 User Datagram Protocol (UDP)

It must be noted that most of the information in this section, has been taken from RFC 768 (Postel 1980), which deals with the User Datagram Protocol.

The UDP protocol corresponds with the Transport layer in Figure 1. UDP is a connectionless protocol. It is mainly used for sending messages that are not guaranteed to be delivered. It has therefore no flow and congestion control. The format of the UDP packet is shown in Figure 5.

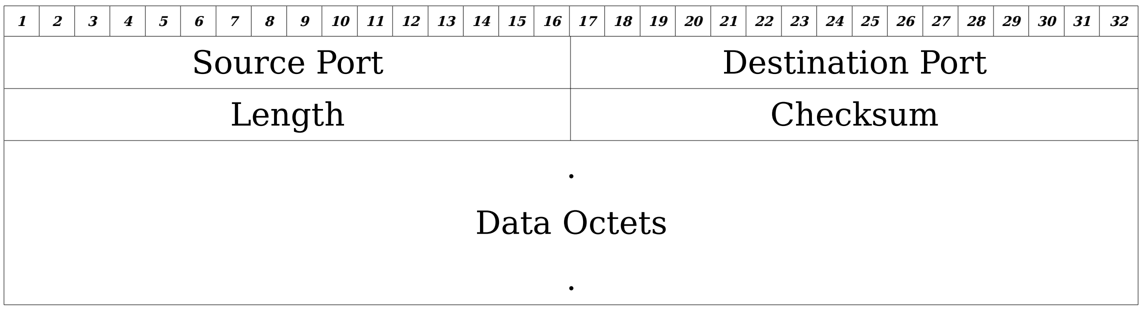


Figure 5 – UDP packet format

Source Port (16 bits): This field is used should a reply be needed to be sent to it. If this field is not needed, it has a value of zero.

Destination Port (16 bits): This specifies the port that the packet is being sent to of the destination address.

Length (16 bits): This field specifies the length of the datagram, including the length of the header and the data, in terms of bytes.

Checksum (16 bits): This is the one's complement of the one's complement of the sum of the pseudo header, the UDP header and the data. The pseudo header prevents against misrouted datagrams. The checksum procedure is the same as in TCP. The format of the pseudo header is shown in Figure 6.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Source Address																															
Destination Address																															
Zero								Protocol								UDP Length															

Figure 6 – UDP Pseudo Header

Data Octets (variable length): This field is padded with zeroes to make it a multiple of two bytes.

UDP is one of the commonly used protocols in port scanning and probing techniques.

1.1.3 Internet Protocol (IP)

The TCP and UDP protocols both rely on the Internet Protocol (IP). The IP protocol corresponds with Internet layer in Figure 1. The reason for having this layer is to allow data to be propagated from one network to another. This concept is often referred to as inter-networking and is the key architectural layer that enables the Internet itself.

The popular IP versions that are currently being used are the Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6).

1.1.3.1 Internet Protocol version 4

It must be noted that most of the information in this section, has been taken from RFC 791 (Postel 1981b), which deals with the Internet protocol.

This version of IP is still used widely around the world. IPv4 has a 32-bit address space, meaning that there are 2^{32} IP addresses that can be assigned world wide.

The purpose of the IP protocol is to move packets between hosts in an interconnected set of networks, taking into account that the packets are limited in size. Therefore, two key functions of the Internet Protocol is addressing and fragmentation.

The higher layers of the TCP/IP stack are responsible for associating the names of the destination with its unique address. These addresses are included in the header of the IP protocol, and the routers and gateways on the network direct the packets to the correct destination.

With Network Address Translation (NAT), a gateway is assigned an IPv4 address. Should a packet be received, the gateway determines which system in the private network requested information from that host, and sends that packet to that system.

Even though NAT is a fast and cost effective solution of giving many computers access to the restricted Internet public address range, it has some problems. One of these problems is that it violates certain IP security mechanisms that rely on IP addresses, since the IP address has effectively been spoofed. As a result of this, many network services will not run through a NAT gateway (Burgess 2006). This is not always effective because many users have the same IPv4 addresses as seen from the Internet, which restricts some users to some servers.

The masquerading of packets is a similar process to NAT, but instead of altering only the IP layer, the TCP layer is also altered, resulting in a new port number being assigned at the gateway. In so doing, two systems behind the NAT gateway will be able to communicate to other systems on the Internet, referring to the same port number. With masquerading, the gateway will also alter the port number of the source to a higher unused port number internally (as well as change the address of the source).

The format of the IPv4 header is discussed in Appendix A.2.

IPv4 has the option of having packets fragmented during transmission. Fragmentation allows larger packets to be divided into smaller size packets (not necessarily equal in size). The primary reason for packet fragmentation is that data might have to traverse many networks to get from source to destination, where the maximum transmission unit (MTU) size may differ from one network to another.

There is, however, a Don't Fragment (DF) flag that can be set in the packet, and should a node not be able to deliver it, it is simply discarded. In this scenario an ICMP error message is returned to the sender. These fragments are controlled by setting the identification field to be the same for the fragments of that packet, therefore preventing the fragments of different packets from being mixed. The More Fragment (MF) flag is set when subsequent fragments are still expected. In the case of the last fragment of a packet, the MF is set to a value of zero.

1.1.4 Internet Control Message Protocol (ICMP)

Most of the information in this section has been taken from RFC 792 (Postel 1981c), which deals with the Internet Control Message Protocol.

ICMP acts as if it is a higher-level protocol than IP, but it is actually part of the IP protocol. It provides feedback about problems that have occurred on the network. It should be noted, however, that these messages are not necessarily sent in the event of an error.

Should an ICMP message be sent, it is indicated in the Protocol field in the IP packet header, which will have a value of 1. There are basically eight different formats for an ICMP packet, each for different types of message.

The 32 bit unused field is not used for all the code values, and should be set zero by the sender. The receiver should ignore this field.

1.1.4.1 Destination Unreachable Message

These messages are sent when a packet is not sent successfully to the destination.

The format of the Destination Unreachable Message ICMP is shown in Figure 7.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Unused																															
. Internet Header + 64 bits of Original Data Datagram .																															

Figure 7 – Destination Unreachable Message ICMP

The fields in the Destination Unreachable Message ICMP has the following functions.

Type (8 bits): The value of this field is 3.

Code (8 bits): The code has the following values for the respective message:

0 = Net unreachable

1 = Host unreachable

2 = Protocol unreachable

3 = Port unreachable

4 = Fragmentation needed and DF set

5 = Source route failed

Codes 0, 1, 4 and 5 are usually received from a gateway. Codes 2 and 3 are usually received from a host.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Internet Header + 64 bits of Data Datagram: Should the higher level protocol use a port number, it is then taken as part of the first 64 data bits of the original packet's data.

1.1.4.2 Echo or Echo Reply Message

In the event that an Echo Message ICMP is sent to a host the host replies with an Echo Reply Message ICMP. Should the host reply with an Echo Reply message, then the Identifier and Sequence Number field should be the same as that of the Echo Message.

The format of the Echo and Echo Reply Message is shown in Figure 8.

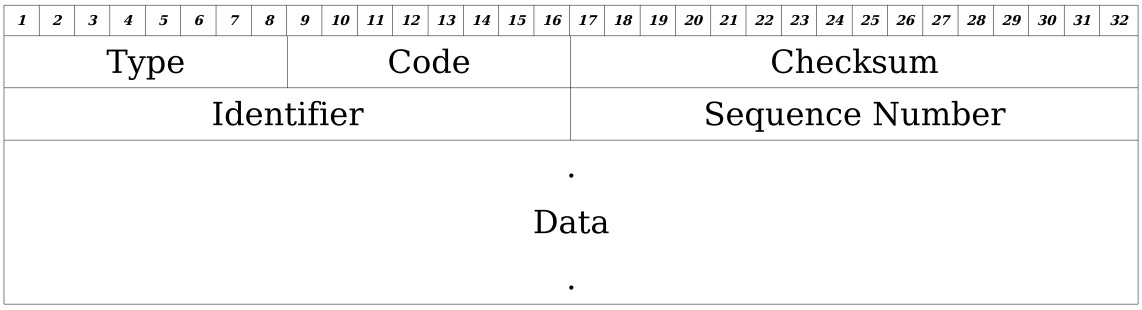


Figure 8 – Echo and Echo Reply Message ICMP

The fields in the Echo and Echo Reply Message ICMP has the following functions.

Type (8 bits): The value of this field is 8 for echo messages and 0 for echo reply messages.

Code (8 bits): The code field has a value of 0 for both type of messages and are usually sent by either a host or a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. Should the total length be odd, then an extra byte of zeroes is added. For the calculation of the checksum, the checksum field is taken to be zero.

Identifier (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Identifier field of the Echo Reply message must be the same as that in the Echo Request message field.

Sequence Number (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Sequence Number field of the Echo Reply message must be the same as that in the Echo Request message field.

The other ICMP formats are discussed in Appendix A.3.

1.1.5 Internet Protocol version 6 (IPv6)

As IPv6 has many security features added to it, further investigation is needed to determine if it will function as a countermeasure for port scanning.

Most of the information in this section has been taken from RFC 2460 (Deering & Hinden 1998), which deals with the Internet protocol version 6.

The changes from IPv4 to IPv6 are that IPv6 has expanded address capabilities, header format specifications, improved support for extensions and options, flow labelling capability and authentication and privacy capabilities. Although IPv6 increases the address space, it is not widely used due to the fact that people are either not affected by this constriction or that most networks use Network Address Translation (NAT).

IPv6 has IP addresses that are 128 bits in length. This results in over 3.4 x 10^{32} addresses that can be assigned to computers.

The formats of the headers and extension headers are discussed in Appendix A.4.

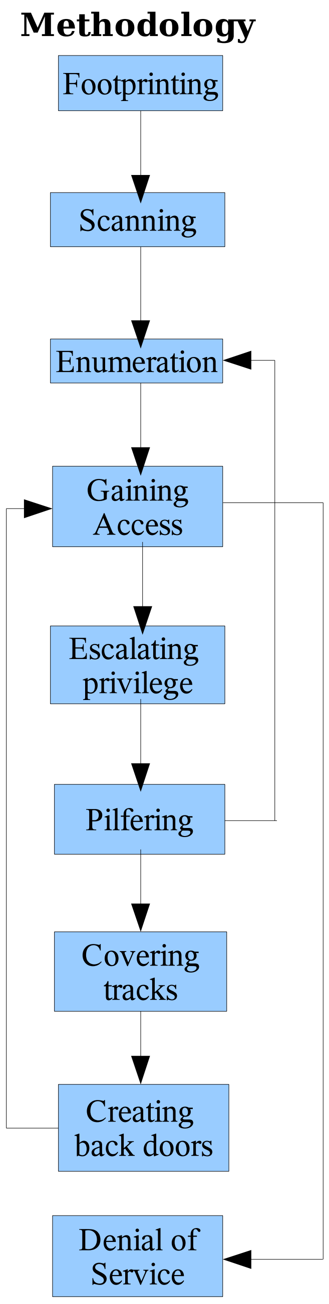
As ICMP is considered as part of the IP, this is different in the case of IPv6. More details of ICMPv6 are discussed in Appendix A.5.

1.1.7 Summary

The information presented in this chapter provided an adequate introduction to network architectures and protocols. Concepts of NAT, IP Masquerading and packet filtering were also introduced. Finally, how the Linux operating system provides these facilities was discussed. This is of interest in the context of the scanning and probing techniques introduced in the following chapter.

1.2. Scanning Techniques

An anatomy of a hack is described by McClure (McClure *et al.* 2005) as shown in Figure 9.



Objective

Target address range, namespace acquisition and gathering information about the target

Identification of services listening. Identifies the most promising point of entry for the attacker

Attacker determines valid user accounts and poorly protected resources

Using the information gathered to attempt to gain access to the system

Instead of just having restricted rights to the system, the attacker seeks to gain access of the complete system

Further information is gathered to gain access to the system as a trusted user

Hiding any evidence from the system administrator that access has been gained by the attacker

Trap doors are set so that the attacker can easily gain full control of the system in the future

As a last resort, the attacker will use readily available exploits to disable the target

Figure 9 – Anatomy Of A Hack

As it can be seen in Figure 9, footprinting is the first step in hacking a system. Footprinting involves the location of the company, postal addresses, administrators telephone numbers and the IP address that has been assigned to them. This information is available on the from the Internet Corporation for Assigned Names and Numbers (ICANN; http://www.icann.org) and Internet Assigned Numbers Authority (IANA). Not much can be done about the public viewing of these pages, apart from restricting access to a target's network details.

Operating System Fingerprinting is part of the Scanning stage. The attacker sends data, using some or other network protocol, to the target system. The response from the target system is monitored and then analysed for useful information such as the operating system, its version, which services are running, and so on.

The Enumeration phase of the hack entails the gathering of information such as user accounts and poorly protected resources. The resulting information gained from the Scanning phase is used during the Enumeration phase. Needless to say, the information gathered in the Enumeration phase can be catastrophic to a computer network. If the Scanning phase is countermeasured to an adequate degree, the attacker would never reach the Enumeration phase.

The first line of defence against scanning, probing, and by extension, operating system fingerprinting, is the network firewall. Firewalls aid the network administrator by filtering network information at any node of the network, including the network gateway to the Internet. The next section will provide a simple introduction to firewalls. The remainder of the chapter discusses the scanning, probing, and fingerprinting techniques used by attackers at the time of writing.

1.2.1 Firewalls

A firewall is a network mechanism that separates systems from the rest of the network by restricting the data that can pass through it, both from the network to the protected system, and from the protected system to the network. For example, the firewall could only allow incoming data to pass through ports that correspond to requests made by systems within the local network. In this sense, firewalls can be utilised as simple packet filters. We refer to this type of firewall as a stateless firewall. Stateful firewalls, however, are more intelligent in that they can filter data at higher levels in the protocol architecture.

Stateful firewalls open the packets and analyse the contents before allowing it to be processed by the system. The fields in the TCP/IP stack are analysed to see whether they are valid, making it possible to know if it is an erroneous packet that was received for potentially malignant or antagonistic purposes.

The most dominant technique of packet filtering in Linux is the IPtables statefull firewall. IPtables gives the user access to packets as they are handed from the NIC to the operating system and, it also provides this access at various stages of propagation through the TCP/IP stack, including just before packets pass out of the operating system and onto the NIC.

The IPtables technology is enabled by providing a set of 'call-back' system calls that give the programmer access to packets just before they pass into the kernel, into the TCP stack implementation, as they propagate up the layers of the TCP/IP stack, down the stack and, finally, just before they are about to leave the kernel.

The result of this software architecture realises a high-level application, called IPtables, that allows system administrators the ability to control network traffic by setting a set of simple conditional rules that control incoming and outgoing network traffic. These rules are simple because, for the network administrator, the firewall is configured by specifying which types of packets are allowed and which are not.

IPtables has a modular architecture, due to the lower-level system-calls it is built with. The various modules are used to add semantics for various types of protocols, routing features such as NAT and IP masquerading, logging and messaging facilities, and, more interestingly, the state of packets in the context of a known connection using a known protocol. If packets don't make sense in the context of a particular 'conversation', an IPtables module can detect this and take appropriate action.

1.2.2 Port Scanning

Recall that the second phase of a hack is the scanning of a system's NIC to see which ports are listening. This could involve TCP and UDP port scanning. It could also involve higher-level analysis such as operating system fingerprinting. There are many types of port scanning techniques, but the following types are the most common TCP scans (McClure *et al.* 2005) and are also used by Network Mapper (Nmap), as described by Nmap's manual pages.

UDP scan: UDP scanning is usually slower and difficult than TCP scans, but some system administrators ignore the monitoring of these ports. Versions scans can be deployed to help differentiate between open ports and filtered ports. The disadvantages for this scan is that it is slow and closed ports send back an ICMP port unreachable error message.

TCP connect scan: This is a three way handshake scan, in which a SYN packet is sent to the target from the attacker. The target replies with a SYN/ACK packet, to which the attacker replies with an ACK packet. This scan is easily detected by the target system. Should this scan be available, it is better to use. The disadvantage, however, is that it takes longer for the scan to be performed. Furthermore, these scans can be detected and logged by the target system. This is one of the oldest types of scan.

TCP SYN scan: This scan is referred to as half-open. This is because a full TCP connection is not established. A SYN packet is sent from the attacker to the target. If a SYN/ACK packet is received from the target, it means that the port is listening. Should a RST/ACK packet be received, it would indicate that the port is not listening. The SYN scan is the most popular scan because it is faster, and stealthy, as a TCP connection is never completely established. This scan is also able to differentiate between open, closed, and filtered ports.

TCP FIN scan: The attacker sends a FIN packet. According to RFC 793 (Postel 1981a), the target system should send a RST packet back to the attacker for a closed port. This scan genrally effective on Unix TCP/IP stacks.

TCP Xmas Tree scan: The attacker sends a packet with the FIN, URG and PUSH flags set (making it look like a Christmas tree).

The PUSH control flag is activated in the packet if the nature of the packet is such that it must be pushed immediately to the receiver. This includes all unsent data that has not been sent from a node before the PUSH packet has been received.

Again, according to RFC 793 (Postel 1981a), a RST packet is sent from the target back to the attacker if the port is closed.

TCP Null scan: A packet is sent from the attacker to the target that has all the flags turned off. According to RFC 793 (Postel 1981a), a RST packet is sent from the target to the attacker if the port is closed.

The advantage of the Null, XMAS and FIN scans is that they can sometimes penetrate certain non-stateful firewalls and packet filtering routers. These scan types are therefore slightly more stealthy. The disadvantage of these scans is that they cannot differentiate between open and filtered ports.

TCP ACK scan: This scan is used to map out firewall rule sets. It can be used to determine if the firewall is a simple packet filter, which only allows established connections (with the ACK bit set) or whether it is a stateful firewall, which performs advanced packet filtering.

TCP Window scan: With this scan it can be determined if ports are open and if they are filtered or not. This is due to the way the TCP window size is reported. Different operating systems report different window sizes. With open ports a positive window size is reported even if a RST packet is also returned. Closed ports are known if a window size of zero is reported. Only a small number of operating systems on the Internet give this report, so this kind of scan cannot be reliable.

TCP RPC scan: This scan is specific to Unix systems, and is used to detect and identify Remote Procedure Call (RPC) ports and the associated services along with their version. This scan can also be used on a MS Windows system on port 445 instead of port 111 on Unix systems.

Maimon Scan (Maimon 1996): Maimon indicates that this is a “Stealth scan” as the SYN flag is not set. It uses two methods. The first method is that a packet with the FIN and ACK flags set are sent. An RST packet will be returned from the target if the port is open, otherwise nothing will be received. The other method is to send an ACK packet, and if the TTL value of the received RST packet is different to that of the other packets received, then this is an indication that the port is open.

For all the scanning methods mentioned above, the target system is able to determine who is performing the scans against it. This is achieved by identifying the system with the same IP address that is scanning the target system on different ports and protocols. Furthermore, this traffic is usually logged. In this event, the system administrator can block any communication with this IP address.

Countermeasures for these scans are available (an example is Psionic PortSentry (www.psionic.com/abacus) from the Abacus project), that detects and responds to the attack. A response to this attack could be to set filtering rules that will not allow the attacker's unique IP address to have access to the system (Postel 1981a). This is not always reliable as the attacker could modify it's IP address for the time period that the scans have been performed, making it possible to continue with the attack. The attacker could also perform a 'denial of service' (DoS) attack with spoofed packets.

The most common and secure countermeasure is to disable all ports and services of the system that are not needed. This countermeasure is probably not the most convenient countermeasure as some open ports will still be needed for effective communication across a network. Another countermeasure that could be implemented is to set filtering rules and allow only selected systems with their specific IP addresses to communicate with it. This will work for situations where a user can dial into the system and then login.

1.2.2.1 A new port scanning proposal

A new type of scanning technique has been invented by Antirez (http://www.kyuzz.org/antirez/), which is known as the basic IPID scan technique. This scanning method is also known as the Stealth, Blind or Idle scan. It works on the concept that there are three computer systems. The Attacker, the Target and the Zombie. The zombie system can be any computer system on the network that is idle. The IPID scan is described in Figure 10.

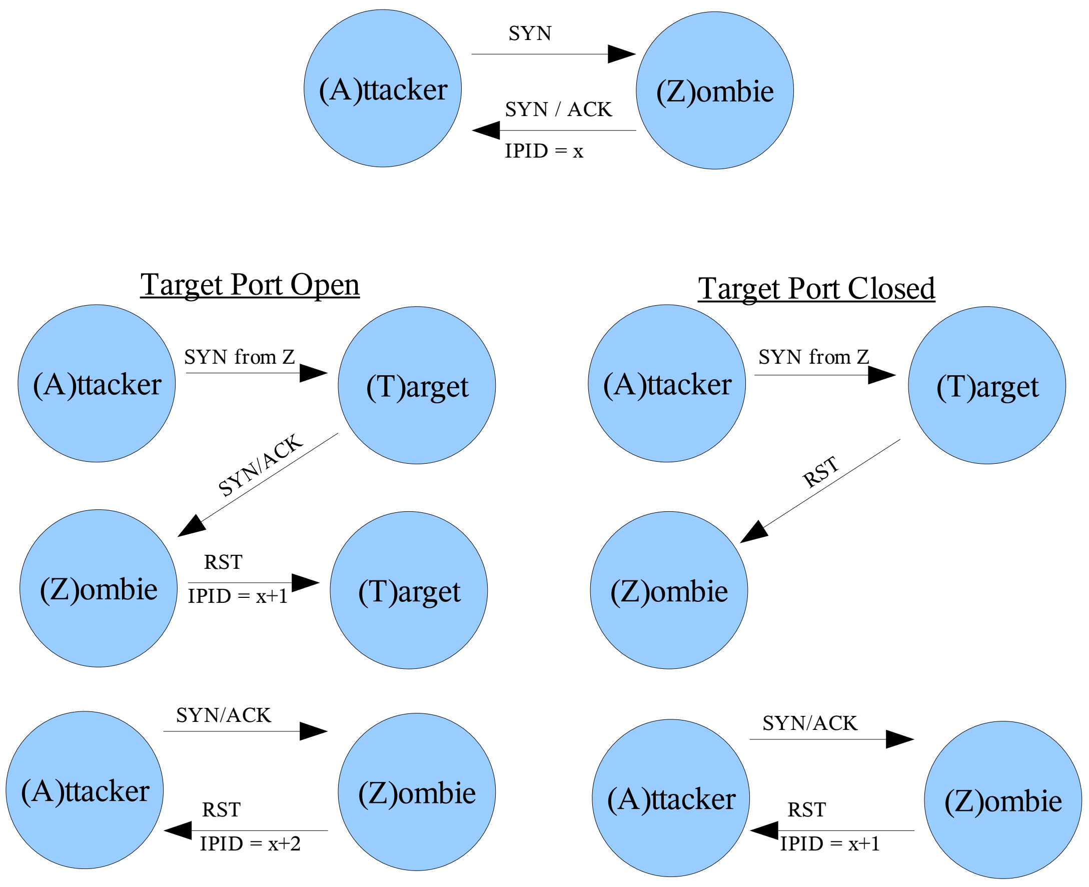


Figure 10 – IPID scan process

The IPID scan operates on the idea that the attacker monitors the IPID (the IP Identification field) of the packets received from the zombie system. The condition that must be observed, is that the zombie system is idle, i.e. it is not actively sending packets over the network.

The attacker sends a SYN packet to the zombie system, which returns a SYN/ACK packet, which has an IPID value of x. The attacker then is faced with two scenarios.

The first scenario occurs when the port is open on the target system. The attacker sends a spoofed packet to the target, in which the destination address is that of the target system, but the source address is that of the zombie system. Since the port is open, the target system sends a SYN/ACK packet to the zombie system. The zombie system did not establish a prior connection to the target system, therefore it will reply with a RST packet which will have an IPID of (x+1). The attacker then again sends a SYN packet to the zombie system, which then replies with a SYN/ACK packet that has an IPID value of (x+2).

The other scenario is if the port is closed on the target system. The attacker sends a spoofed packet to the target, which has the destination address of the target system, but the source address is that of the zombie system. As the port is closed, the target system sends a RST packet to the zombie system, to which the zombie system does not react. The attacker then sends a SYN packet to the zombie system, which then replies with a SYN/ACK packet that has an IPID value of (x+1).

The attacker knows if the specific port of the target system is open if the difference in IPIDs is 2, otherwise it is closed should the difference of the IPIDs be 1.

The advantage of the IPID scan is that the target system has no records that the attacker is trying to do a port scan on it. The target system thus has only records of the zombie system scanning its ports, and therefore can set filtering rules to block any communication with the zombie system.

Another advantage is that the attacker can use a zombie system that the target system trusts. In this case, the target system won't mind if scans are done from the trusted system, allowing the attacker to do a thorough scan without being interrupted.

This well designed scanning method so impressed Gordon Lyon, also known as Fyodor (a hacker) that he programmed this scanning method in his popular program Network Mapper, also commonly known as Nmap.

The IPID scan does not work on zombie systems running OpenBSD and Linux kernel 2.4.x, as the IPID values are sequential. This is discussed in section 1.*2.4 Operating System (OS) Fingerprinting*.

According to Fyodor (http://www.insecure.org/nmap/idlescan.html) there are challenges with the IPID scan. Scanning the ports one at a time could take a lot of time. Another challenge is that, should the zombie be a non-idle host, then this will make it difficult to scan for open ports. The solution is to either use another system as the zombie system, or as Fyodor has done in Nmap, to rescan for open ports should the IPID increment be greater than two for each port.

Another challenge for IPID scanning is egress filtering done by the Internet Service Provider or ISP (e.g. the ISP checks to see if the packet being sent from it actually comes from a valid system in its network). To bypass this, one could either try another ISP, or use IP tunnelling. Another method is to use a zombie system that is in the same network resulting in the ISP sending the spoofed packet.

Since the attacker does not receive any packets from the target, it is impossible to use IPID scanning for operating system fingerprinting (as shown in section *2.3.2.2 Xprobe2's Results*) of the target system.

1.2.3 Other TCP/IP Attacks

As a result of the previous discussions, the worst type of attack that uses the TCP/IP stack is port scanning. Therefore, some attacks, described by

Dr. K (Dr. K 2000), that use the TCP/IP stack are discussed next.

1.2.3.1 IP Spoofing (IP Hijack)

This attack involves the attacker, a target and a trusted host that the target is communicating with. The attacker SYN-floods the trusted host, possibly using an illegitimate source address. At the same time the attacker sends a SYN packet to the target. The target replies with a SYN/ACK packet with a sequence number, making it possible for the attacker to guess of a possible sequence number that the target is using to communicate with the trusted host. The attacker will adjust the packets sent to the target so that the host will believe the packets are coming from the trusted host.

The attacker then sends a spoofed SYN packet to the target with the source address of the trusted host. The target replies to the trusted host with a SYN/ACK packet, but the trusted host does not receive it as it's buffer is overflowing with the SYN-flooding.

Thereafter the attacker sends an ACK packet to the target with the source address of the trusted host and a sequence number previously guessed. If the sequence number is correctly guessed, then the attacker can communicate with the target and penetrate it. Similar techniques can be used for packet alteration as described in section *1.2.2 Port Scanning*.

1.2.3.2 Address Resolution Protocol (ARP) Spoofing

ARP is a protocol used to identify an unknown IP address with a system's MAC address. ARP spoofing is the process where an attacker will send spoofed ARP messages on the network that contain the false MAC address. In this way the packets that were intended for a system are sent to the attacker's system for analysis. ARP spoofing is also used for Denial of Service (DoS) attacks, if the host becomes unreachable, and can only be done on local LAN segments.

The technique of ARP spoofing makes it possible for a man-in-the-middle attack, in which the attacker acts as a router and forwards packets between systems. This gives the attacker the opportunity to either analyse the packets being sent, or to alter them and send inaccurate packets to the destination system.

1.2.3.3 ICMP Attacks

Ping Flood: A program such as Packet InterNet Gopher (PING) is used (PING is part of "iputils" package and the latest versions are available in source form for anonymous ftp ftp://ftp.inr.ac.ru/ip-rout-ing/iputils-current.tar.gz.) The UDP protocol is mainly used for this attack, even though some versions of PING can send TCP packets. Due to the fact that the TCP/IP stack has higher priorities than that of user programs, the user programs will start to run slowly. This is because the processor is spending most of it's time on the incoming ICMP Echo packets and returning ICMP Echo Reply packets. This is more effective if a number of computers attack a single host at the same time. Ping flooding will also reduce the available network bandwidth to a system, resulting in the valid users of that system to experience very slow access.

Oversized Packets (Ping of Death): When a packet is larger than 65536 bytes , and after the fragments have been combined, the buffer overflows. With older OS such as MS Windows 3.1, MS Windows 3.11 and DOS, the Ping of Death can be successfully achieved with packet sizes of 7999 bytes. This can cause the system to reboot, shutdown, fail, or the kernel to start panicking.

Destination Unreachable Message to "Nuke" network connection: The target has an established connection with a server. The attacker then sends a Destination Unreachable ICMP message to the target that has the same source address as that of the server. The target is disconnected but this gives the attacker an opportunity to connect to the server as the target.

Traceroute: This is not really an attack, but it gives the attacker the opportunity to discover the route that the packets are travelling. Traceroute tracks the route that packets travel across a TCP/IP network to a specified host. Since a Time Exceed ICMP packet is sent to the attacker when the TTL value has reached zero, the attacker starts sending a packet with a small value in the TTL field. The attacker receives these messages, with the address of the node that sent it. Every time the attacker increments the TTL value, the next node sends the message. If the packet that the attacker is sending has a port number that does not exist then, as soon as the packet has reached its destination, the target will return a Port Unreachable message to the attacker. Since traceroute relies on the TTL value in the IP protocol, other protocols that travel with the IP protocol packet could also be used to route the packet path.

Bypass Firewalls: Some firewalls (depending on the configuration) don't allow ICMP messages through, but some are allowed through for the network to perform properly. A program like Simple Nomad's “icmpenum” uses the Timestamp Request and ICMP Information messages to be able to map the network behind the firewall. This will only be possible if the firewalls are configured to allow Timestamp Request or ICMP Information messages through.

Project Loki (daemon9 1996): The technology developed in this project stores data inside Echo messages. It allows attackers to retrieve information from a target that has been compromised before, with the request of these messages and without the permission of the user of the target system. The project's objection was initially to allow traffic between two systems without noticing the flow of traffic between them, but it can be a tool used by an attacker. This tunnelling effect might also allow packets to bypass the firewall as described above.

1.2.4 Operating System (OS) Fingerprinting

Operating system fingerprinting is the process of detecting the operating system of a remote target. It is done by probing – sending different “formats” of packets and analysing the packets received is response. This technique is considered to be part of the scanning process because it the same technical process as port scanning techniques do.

It must be noted that most of this information is gathered from Fyodor's paper (Fyodor 1998) as well as from Ofir Arkin's paper (Arkin 2001a).

The reason why detection of the OS of a remote system is necessary is that when an attack is made on a system (whether it is a hacker or a system administrator trying to audit how secure the system is), the attacker might only have one chance of doing so. The daemon might crash and, in this way, notify the administrator that somebody is trying to penetrate the system. This usually only occurs when exploit code is being used.

The way that the detection used to be performed in the past was that a telnet connection (or a similar type) was established with the system, and the system would display a banner of the operating system running. This was not an accurate way of detecting the operating system, as the system administrators could either disable the display of the banner, or have the banner display OS information that was untrue.

As mentioned before, the new way of doing the OS detection is by analysing the packets sent back from the target system. Thus it is possible to not only know what OS is running on the system, but the version as well.

According to Arkin, the techniques used by Nmap are not sufficient and therefore ICMP based OS detection is better. Many OSs have not changed the TCP / IP stack in the past versions and service packs, therefore making it difficult to differentiate between them. This is where Xprobe2 (http://www.sys-security.com) uses a different approach to detect the OS. Xprobe2 is also able to often send and receive ICMP messages as the firewalls don't always block them. This gives the attacker the opportunity to view packets sent from a system behind a firewall. This makes Xprobe2 different to Nmap, even though Fyodor has implemented these type of scans in the OS fingerprinting in the later versions (as investigated in version 4.00) of Nmap. These techniques are also discussed.

The following techniques are used to determine the OS on the system.

The FIN probe: A packet is sent to a system that either has the FIN control flag set, or where the SYN and ACK control flags are not set. According to RFC 793 (Postel 1981a), these packets must be ignored. Packages such as MS Windows, BSDI, CISCO, HP/UX, MVS, and IRIX send a RST packet in return.

IPID sampling: Most OSs increment the IPIDs for each consecutive packet they send. OpenBSD use random values for the IPID while Linux version 2.4.x and up use an IPID value of zero as well as having the DF control flag set. MS Windows increments the IPID with a value of 256.

Don't Fragment (DF) bit: Some OSs set this bit while others don't.

TCP ISN sampling: This looks at the initial sequence number. The different OSs can be classified into the following categories.

Table 1 – ISN values for different OS

ISN	**Operating System**
64k	Old Unix systems
Random increments	Newer versions of Solaris, IRIX, FreeBSD, Digital UNIX, Cray, etc.
True Random	Linux 2.x, OpenVMS, newer AIX, etc.
Fixed increment (Time dependant)	MS Windows
Constant (same ISN)	3Com hubs, Apple Laserwriter printers

TCP Initial Window: The monitoring of the window size of the returned packets is done. This window size is relatively constant for each OS. Interestingly enough, with MS Windows NT the window size used is the same as that of OpenBSD and FreeBSD.

ACK value: Some OSs differ as to how this value is set. If a FIN/PSH/URG packet is sent, then most OSs will set the ACK value to that of the ISN. Unlike MS Windows and printers that don't follow RFC 793, this field will have an incremented value of the sequence number.

ICMP Error Message Quenching: Some OSs limit the number of ICMP error messages that are sent within a second.

ICMP Message Quoting: Most OSs send only the IP header and 8 bytes back, while Sun Solaris returns one extra byte but Linux sends even more bytes back. Such OSs are HPUX 11.x, MacOS 7.x-9.x and Foundary Switches.

ICMP error messages echoing integrity: Systems have to send back part of the original message with the port unreachable error. Often the headers are messed up. AIX and BSDI send back the total header size to be

20 bytes too high, while other OSs have a total header size of 20 bytes less.

AIX and FreeBSD send back a checksum that is either not valid or with a value of zero. Many OSs miscalculate the UDP header checksum or set it to zero.

Some OSs don't echo the 3-bit flags and Offset field values correctly.

A program such as Nmap does nine different tests on a system and depending on the results it receives, it determines the OS.

Type of Service (TOS): This field is usually set to zero for all OSs. Linux has the precedence bits of the TOS field set to 0xC0. ICMP error messages are always sent with the default TOS value of 0x0000. The ICMP echo reply message should have the same TOS value as the ICMP request message.

TCP Options: The following options are usually sent in all packets used for probing:

Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops;

FreeBSD support all of the above options, while Linux 2.0.X only supports a few. Linux version 2.1.X and above support all of these options. The values of the options that are returned differ between OSs. Even if the values are the same, the order differs. Solaris will return:

<no op><no op><timestamp><no op><window scale><echoed MSS>

while Linux 2.1.122 will return:

<echoed MSS><no op><no op><timestamp><no op><window scale>

Exploit Chronology: MS Windows 95, 98 or NT don't have any differences in the TCP stack as the TCP stack has not changed during the versions. A way to be able to determine the version of the Microsoft OS, is to do attacks on the system (such as Ping of Death etc.) and carry on with nastier attacks, until the system has crashed. Each version has patched some of these loopholes.

Another way is to use a method used by Xprobe2 that monitors the fields and reactions that the OS performs when it receives an ICMP message.

SYN Flood Resistance: Some OS are not resistant to SYN floods, as they can only handle 8 packets at a time. This is a useful (but not a friendly) tool to determine the OSs of a system.

Time-To-Live (TTL): This field has a different value for ICMP query messages and for ICMP query replies.

By combining the above results of the scans performed on a system it can be determined what OS and version the system is running.

Xprobe2 uses an Optical Character Recognition (OCR) approach (Arkin *et al.* 2003), which makes use of a matrix based OS fingerprinting. Xprobe2 monitors if the OS on the target system responds to ICMP Echo Request, ICMP Timestamp Request, ICMP Address Mask Request and ICMP Information Request with different bits set. The fields in the received packets are then observed. Some of the fields are:

- IP TTL field

- The value in the code field
- Precedence bits
- TOS field
- TOS byte unused bit
- DF bit
- IP packet total length
- IPID
- IP Header Checksum
- UDP Checksum
- ICMP ISN
- Content offset from the ICMP header

The adding the statistical scores of the different tests performed gives a total score that is possible to determine the OS running on the target system. The tests performed by Xprobe2 are similar to those of Nmap, except that Xprobe2 is only reliable on the reception of ICMP packets.

The approach Xprobe2 uses differentiates between Linux kernel versions by exploiting the fact that versions 2.0.x use 64 as the initial TTL field, while versions 2.2.x and 2.4.x use 255 for this field. Linux with a kernel version of 2.4.x and higher has an IPID value of zero.

There are other less popular fingerprinting tools available that have been investigated.

QueSO is an OS fingerprinting tool that uses some of the techniques that Nmap uses. It was created by the Apostols and can be found at www.apostols.org/projectz/queso/. Netcraft (www.netcraft.com) is a website that surveys webserver platforms as a primary role, and consequently also does some basic OS fingerprinting. It fingerprints in a similar basis as Nmap but it states that the OS fingerprinting might be inaccurate if the system has “changed the default configuration of their TCP/IP stack.”

Nessus (www.nessus.org) is a program specifically designed for network auditing. It has the ability to do OS fingerprinting, but it uses a plug-in from Nmap. From this, it is concluded that Nessus uses the same scanning techniques that Nmap does.

P0f (Zalewski 2004) is a passive fingerprinting tool. It does similar checks as Nmap does, but instead of sending packets to the target to analyse the packets, it monitors the packets that are being received with either an incoming or outgoing connection. This type of scanning has the disadvantage that the attacker has to wait for a legitimate connection to be established before the OS fingerprinting can be performed. The advantage it has compared to Nmap, is that P0f can fingerprint OS behind a firewall or NAT. Another advantage is that it has an advanced masquerade detection. The way that P0f is able to perform this type of detection is by recording the changes in the TCP/IP packets that come from the same IP address. Zalewski states though that most of the detections and identifications performed by P0f are not accurate and are there-for purely "*amusement value*". He also states that "*P0f will never be as precise as Nmap*", which is another disadvantage.

Veysset *et. al.* created the program Ring, from which Tod Beardsley (Beardsley 2003) created Snacktime. A patch for Nmap has been implemented that adds this scanning method. These programs operate on the basis of determining the retransmission of packets for the TCP handshake. Three methods are used for this fingerprinting.

The first method is that the attacker has a firewall activated on his machine to drop all SYN/ACK packets when the port is open. The attacker sends a SYN packet to the target machine, to which the target replies with a SYN/ACK packet. As the target system is not receiving an acknowledgement of the reception of the SYN/ACK packet, it retries to send it for a limited number of times, with a different delay between the retries.

The second method is that the attacker sends a SYN packet to the target system. The target replies with a SYN/ACK packet. The attacker replies with an ACK packet. The firewall of the attacker's system is enabled and then a FIN/ACK packet is sent to the target. The target machine replies with a FIN/ACK packet, which is dropped at the attacker's system. The attacker then monitors the number of limited packets and the delays between the retries sent by the target system.

The third method has not been fully implemented in the Ring program yet. The attacker sends a SYN packet to the target, to which the target replies with a SYN/ACK packet. The attacker acknowledges and then sends a PSH/ACK packet to the target system. The target system acknowledges and then sends a PSH/ACK packet to the attacker. The attacker then enables the firewall to block all FIN/ACK packets and then acknowledges with an

ACK packet. The target system then sends FIN/ACK packets that are limited in retries and with different delays between them to the attacker's system.

These retries and delays are compared with a database to determine the OS being used.

From the above investigated programs, it can be determined that the different techniques used for OS fingerprinting are represented by Nmap and Xprobe2. It is believed that any countermeasures for these fingerprinting tools will be sufficient for the other tools available.

1.2.5 Summary

Different port scanning techniques, TCP/IP and ICMP attacks were investigated. The way that OS fingerprinting is achieved was shown and it was also determined which fingerprinting tools will be sufficient to do the fingerprinting experiments.

Chapter 2: Tests and Experiments

Various tests were performed to investigate the efficiency of the different port scans and operating system fingerprinting. These tests were performed in situations when a firewall was either disabled, or enabled , with a standard Linux distribution – Suse 10.1.

2.1. Practical Investigation with IPv4

2.1.1 IPID scan

A practical investigation is needed to confirm that Linux kernel version 2.4.x and above is immune to the IPID scan. These tests have been performed on Linux with a kernel version 2.4.x (Arkin 2001b). From Arkin's tests the IPID remained zero. This investigation should determine what would happen if the zombie system runs on a Microsoft Windows platform, or a Linux kernel smaller than version 2.4, irrespective of whether the target is running a Linux kernel version greater than 2.4. It is known that Microsoft Windows increments the IPID by a factor of 256. (Fyodor 1998).

The firewall uses the IPtables rules specified in Appendix B.2. From these rules it can be seen that all broadcast and multicast packets are dropped. The TCP packets are rejected with a TCP RST packet, except those that are used for SSH. UDP packets are rejected with a Port Unreachable ICMP while all other types of packets are rejected with a Protocol Unreachable ICMP.

There are various scenarios that have been considered. Four scenarios were chosen. The first two scenarios investigate and validate the effectiveness of IPID scans when a Linux system plays the role of zombie system, with or without a firewall. The third scenario solidifies the results by using a different Linux distribution to the first two scenarios. The last scenario is to test the IPID scan when a Microsoft server plays the role of zombie system and Linux plays the role of the target system.

Three computer systems were set up to demonstrate the IPID scan. Professional Hacker's Linux Assault Kit (PHLAK) was used as it contains a large variety of hacking tools. PHLAK can be found at http://phlak.org. Microsoft Windows XP Professional with service pack 2 was used as this is one of the common Microsoft operating systems used at the time of writing, and it is not immune to Nmap when it plays the role of zombie system. SUSE 10.1 was used as this is a stand-alone Linux system with a kernel version that is greater than 2.4.x. The computers configured as follows:

Computer A:

IP Address: 10.0.0.5

Operating System: Windows XP Professional with Service Pack 2

Computer B:

IP Address: 10.0.0.6

Operating System: Professional Hacker's Linux Assault Kit (PHLAK) running a Linux kernel version 2.6.9

Computer C:

IP Address: 10.0.0.7

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

2.1.1.1 Scenario 1 – Confirming if Linux 2.6.x is immune to Nmap when it is a zombie system

In this scenario, computers A, B and C are considered to be the target, zombie and attacker systems respectively. Nmap was executed on computer C to scan computer A for open ports via computer B.

The following output was found:

```
linux:/ # nmap -P0 -p- -sI 10.0.0.6 10.0.0.5

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-06-20 09:18 SAST
Idlescan using zombie 10.0.0.6 (10.0.0.6:80); Class: Incremental
Interesting ports on 10.0.0.5:
(The 65530 ports scanned but not shown below are in state: closed|filtered)
PORT     STATE SERVICE
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS
5000/tcp open  UPnP
MAC Address: 00:50:22:A4:46:FE (Zonet Technology)

Nmap finished: 1 IP address (1 host up) scanned in 256.225 seconds
```

As seen from the results, five open ports were detected in the scan. From these results, it is clear that a Linux kernel version 2.6.9 is not immune to Nmap when acting as a zombie system. It also indicates that Microsoft Windows XP Professional is not immune to Nmap when acting as a target system.

2.1.1.2 Scenario 2 – Confirming if Linux 2.6.x is immune to Nmap when it is a zombie system and it has a firewall

In this scenario, computers A, B and C are considered to be the target, attacker, and zombie systems, respectively. Nmap was executed on computer B to scan computer A (which has the firewall disabled) for open ports via computer C.

The following output was found:

```
linux:/# nmap -P0 -p- -sI 10.0.0.7 10.0.0.5

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-06-20 09:41 EDT
Idlescan using zombie 10.0.0.7 (10.0.0.6) port 80 cannot be used because IPID sequencability class is: All zeros. Try another proxy.
QUITTING!
```

This output shown above indicates that a Linux kernel version 2.6.16 and higher is immune to being a zombie system.

The same scenario was considered, but this time the firewall was enabled on computer C.

The following output was found:

```
linux:/# nmap -P0 -p- -sI 10.0.0.7 10.0.0.5

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-06-20 09:49 EDT
Idlescan using zombie 10.0.0.7 (10.0.0.6) port 80 cannot be used because it has
```

```
not returned any of our probes – perhaps it is down or firewalled.
QUITTING!
```

From the results shown above the observation is that a zombie system running Linux kernel version 2.6.16, with a firewall enabled, is not able to find any open ports on the target system.

2.1.1.3 Scenario 3 - Confirming if another Linux OS running kernel 2.6.x is immune to Nmap when it is a zombie system

In this scenario, computers A, B and C are considered to be the target, attacker and zombie systems respectively. Nmap was executed on computer B to scan computer A for open ports via computer C. Computer C's Operating system was changed to PHLAK, running a Linux kernel version 2.6.9. This was done so that tests could be performed on a different Linux kernel version.

The following output was found:

```
linux:/# nmap –vv P0 -p- -sI 10.0.0.5 10.0.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-06-20 10:01 EDT
Idlescan using zombie 10.0.0.7 (10.0.0.6:80); Class: Incremental
Initiating Idlescan against 10.0.0.7
Discovered open port 25/tcp on 10.0.0.7
Discovered open port 21/tcp on 10.0.0.7
Idlescan is unable to obtain meaningful results from proxy 10.0.0.5 (10.0.0.5). I'm sorry it didn't work out.
QUITTING!
```

In this scenario, the very verbose mode of Nmap was considered, and even though Nmap considered the scan not to be meaningful, two open ports were found. This again indicates that a system running Linux kernel version 2.6.9 cannot be considered immune to Nmap because two open ports were found.

2.1.1.4 Scenario 4 - Confirming if Linux 2.6.x is immune to Nmap when it is a target system

In this scenario, computers A, B and C are considered to be the zombie, attacker and target systems respectively. Nmap was executed on computer B to scan computer C (with the firewall disabled) for open port via computer A.

The following output was found:

```
linux:/# nmap –vv P0 -p- -sI 10.0.0.5 10.0.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-06-20 10:01 EDT
Idlescan using zombie 10.0.0.7 (10.0.0.6:80); Class: Incremental
Initiating Idlescan against 10.0.0.7
Discovered open port 22/tcp on 10.0.0.7
WARNING: Idlescan has erroneously detected phantom ports – is the proxy 10.0.0.5 (10.0.0.5) really idle?

(Continues and Repeats the scanning process)
```

The results shown above suggest that a system running Linux 2.6.16 is not immune as a target to Nmap if the zombie system is running on Microsoft Windows XP Professional.

The same scenario is considered, but this time the firewall on the target system is enabled.

The following output was found:

```
linux:/# nmap -P0 -p- -sI 10.0.0.5 10.0.0.7

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2006-06-20 10:01 EDT
Idlescan using zombie 10.0.0.7 (10.0.0.6:80); Class: Incremental
(Continues and Repeats scan as no reply is received)
```

From the results of this scan, no open ports were found. The packet analysing tool Ethereal (found at http://www.ethereal.com) was executed the same time when the scan was done, and it was observed that no response was received from the target system.

2.1.1.5 Microsoft Windows XP Professional IPID incremental

While these computer systems were set up, it was decided to monitor the packets being sent from the Microsoft Windows XP Professional system. HPING(1,2,3), which is a TCP pinging program developed by Antirez (http://www.kyuzz.org/antirez/), was used to monitor the response from the Microsoft Windows XP Professional system. While Ethereal was running at the same time, it was interesting to see that the IPID's of the packets that were received from the Microsoft operating system were incrementing by a value of one, and not 256 as stated before.

2.1.2 Other port scanning techniques

Other port scanning techniques were investigated, again taking into consideration that the standard firewall that came with SUSE 10.1 was either enabled or disabled. The system's settings and the outputs of the scans are displayed in Appendix C.1.

2.1.2.1 SYN Scan

As seen from the outputs in Appendix C.1, Nmap was able to detect that the SSH port was open, irrespective of whether the firewall was enabled or not. Without the firewall enabled, ports 111 (rpcbind) and 631 (ipp) were also detected as being open, while port 113 was detected as closed for authentication when the firewall was enabled. Recall that open ports are useful for an attacker as these could be vulnerable areas of penetration.

2.1.2.2 UDP Scan

As seen by the output in Appendix C.1, with the firewall disabled, Nmap saw four ports open / filtered on system B, being ports 68 (dhcpc), 111 (rpcbind), 631 (ipp) and 1024 (udp). Packets were dropped in the transmission so Nmap had to increase the delay between the packets sent to 800 ms. This resulted in the time period for the scan to be 1 490 seconds.

Once the firewall was enabled Nmap thought that all the ports were open / filtered. The time period for the scan was shorter with a time of 46 seconds.

2.1.2.3 TCP Scan

As seen from the results from the output in Appendix C.1, the ports that were found open when the firewall was disabled were ports 22 (SSH), 111 (rpcbind) and 631 (ipp). However when the firewall was enabled, ports 22 (SSH) and 113 (closed for authentication) were found by Nmap.

2.1.2.4 Null Scan

The outputs in Appendix C.1 indicate that Nmap saw ports 22 (SSH), 111 (rpcbind) and 631 (ipp) to be open / filtered when the firewall was disabled. However with the firewall enabled Nmap assumed that all 1 672 ports scanned were open / filtered.

2.1.2.5 ACK scan

It is of interest to see from the output in Appendix C.1 results that with the firewall disabled all 1 672 ports scanned were seen unfiltered, while with the firewall enabled, ports 22 (SSH) and 113 (authentication) were seen as unfiltered.

2.1.2.6 FIN scan

From the results in Appendix C.1, it is evident that with the firewall disabled, ports 22 (SSH), 111 (rpcbind) and 631 (ipp) are seen as open / filtered, while with the firewall enabled all the 1 672 scanned ports are seen by Nmap as open / filtered.

2.1.2.7 Window scan

Once again the outputs in Appendix C.1 were interesting. With the firewall disabled Nmap reported that all 1 672 ports scanned were closed, but with the firewall enabled, it reported that ports 22 (SSH) and 113 (authentication) were closed.

2.1.2.8 Xmas scan

As observed from the outputs in Appendix C.1, with the firewall disabled ports 22 (SSH), 111 (rpcbind) and 631 (ipp) were seen as open / filtered, but with the firewall disabled, all 1672 scanned ports are reported as open / filtered.

2.1.2.9 TCP Maimon scan

As observed from the outputs in Appendix C.1, it is noted that with the firewall disabled, all 1 672 scanned ports are seen as closed, while with the firewall enabled, port 22 (SSH) is seen by Nmap as closed.

2.1.2.10 Protocol scan

The IP protocol scan is one that Nmap used to determine the protocols that the target system is compatible with. It is not really a port scanning technique, but it needs to be investigated to determine if any information can be gained from this scan.

Resulting from the outputs in Appendix C.1 it can be seen that without the firewall enabled protocol 1 (icmp) and 6 (tcp) are available, protocols 2 (igmp) and 41 (ipv6) are open / filtered and protocol 17 (udp) is filtered. With the filter enabled only protocol 1 (icmp) is seen by Nmap.

From the overall outputs given by Nmap, it is concluded that the firewall does not always hide which ports are open or closed. Often Nmap reported that a specific port was closed (referring specifically to port 22 for SSH that was enabled for test purposes) which could give an attacker the idea that the port could actually be open. Should a firewall be the solution for port scanning? More development is needed to improve it. The results of these experiments have been summarised in Table 2.

Table 2 - Comparison of the different scanning techniques

Scan	***Firewall Disabled***	***Firewall Enabled***
SYN	22, SSH-open 111, rpcbind-open 631, ipp-open	22, SSH-open 111, auth-closed
UDP	68, dhcpc-open/filtered 111, rpcbind-open/filtered 631, ipp-open/filtered 1024, udp-open/filtered	All 1672 ports open/filtered
TCP	22, SSH-open 111, rpcbind-open 631, ipp-open	22, SSH-open 113, auth-closed
Null	22, SSH-open/filtered 111, rpcbind-open/filtered 631, ipp-open/filtered	All 1672 ports open/filtered
ACK	All 1672 ports unfiltered	22, SSH-unfiltered 113, auth-unfiltered
FIN	22, SSH-open/filtered 111, rpcbind-open/filtered 631, ipp-open/filtered	All 1672 ports open/filtered
Window	All 1672 ports closed	22, SSH-closed 113, auth-closed
Xmas	22, SSH-open/filtered 111, rpcbind-open/filtered 631,ipp-open/filtered	All 1672 ports open/filtered
Maimon	All 1672 ports closed	22, SSH-closed
Protocol	1, ICMP-open 2, IGMP-open/filtered 6, TCP-open 17, UDP-filtered 41, Ipv6-open/filtered/	1, ICMP-open

These results are discussed further in *section 2.2 – Deducible Observations.*

2.1.3 OS Detection

It should also be taken into consideration whether the firewall affects the detection of the operating system. Recall that it is crucial for the attacker to know the operating system of the target to be able to compromise a the target entirely.

A comparison of OS detection is done using Nmap and Xprobe2.

Two computer systems were set up to demonstrate the scans. The computers used had the following configurations:

Computer A:

IP Address: 10.0.0.5

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Computer B:

IP Address: 10.0.0.6

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Opened ports/services: port 22 / SSH

System A was used to do the scans on system B, with the scenario that system B has the firewall disabled and enabled respectively.

The system configurations and the outputs are displayed in Appendix C.2.

2.1.3.1 Using Nmap

As seen from the outputs in Appendix C.2, Nmap was able to detect that the SSH port was open, irrespective of whether the firewall was enabled or not. Without the firewall enabled, ports 111 (rpcbind) and 631 (ipp) were also detected, while port 113 was detected as closed for authentication when the firewall was enabled.

In both cases the Linux version was detected (not exactly to version 2.6.16.13-4, using Nmap version 4.00), even though when the firewall was enabled on system B it caused Nmap to determine a greater variety of versions.

In comparison when the firewall was enabled, the time period for the scan was more than double compared to when the firewall was disabled.

From the outputs in Appendix C.2 it can be observed that the firewall enabled on a system does help against SYN scanning as well as for OS detection, but not in great depth. Nmap was still able to detect that the SSH port was open (which is probably one of the most promising ports to attack from the attacker's perspective especially if the version of SSH are between 1.2.24 and 1.2.31 (McClure *et. al.* 2005 pgs 255 - 258), as well as being able to determine that it was a Linux kernel running on the target.

2.1.3.2 Using Xprobe2

From the outputs in Appendix C.2 it is observed that when the firewall was disabled, Xprobe2 detected the OS that was running on system B was Linux with a kernel version of 2.4.22 or higher. Xprobe2 was a bit confused when the firewall was enabled. Xprobe2 thought the OS was Foundry Networks IronWare Version 03.0.01eTc1 on its primary guess, and then it couldn't decide between a Linux kernel of version 2.4.21 and higher, or Foundry Networks IronWare. There were more guesses that the OS was Foundry Networks IronWare.

Ethereal was used as well, and the difference in packets sent between the two systems when the firewall was enabled or disabled were investigated. The following differences were found:

- With the firewall disabled, a Destination Unreachable ICMP was returned from system B, as system A sent the packet to port 65534, which was obviously closed. When the firewall was enabled, the ICMP message was not sent from system B.

- System A sent a couple of TCP SYN packet to system B. In the situation that the firewall was disabled, RST / ACK packets were returned. When the firewall was enabled, system B simply dropped these TCP SYN packets and did not send any ICMP packet back to system A.

It was also observed that the TTL field in the TCP had a value of 64. The abstracts of the Ethereal outputs can be found in Appendix H.

2.2. Realisation of Preliminary Countermeasures

Port scanning techniques were first investigated with the use of IPv4. A comparison of the scans is shown in Table 2.

The SYN scan is found to be powerful enough to detect port 22 (SSH) as open when the firewall is enabled. This is the same as the TCP scan, as the SYN scan is part of the TCP three-way handshake.

The ACK scan showed that all the ports are unfiltered when the firewall was disabled, but port 22 (SSH) was seen as listening and unfiltered when the firewall was enabled.

When the Window scan was performed, it was observed that all the ports were seen as closed when the firewall was disabled, but port 22 (SSH) was seen as closed when the firewall was enabled. The problem here is that even though Nmap detected and showed specifically that this port was closed, it could be an indication to the attacker that a firewall is present, or that the port is currently closed, but could be opened in the future.

A similar situation is observed in the Maimon scan. With the firewall disabled, all the scanned ports are seen as closed, but port 22 (SSH) is seen as closed when the firewall is enabled.

From these observations it can be concluded that the most powerful scans of Nmap are the SYN, TCP, ACK, Window and Maimon scans, as these will show available ports even if the pre-installed firewall of Suse 10.1 is activated in the Linux system. A weighted matrix was generated from these results and a graph of these results are shown in Figure 11.

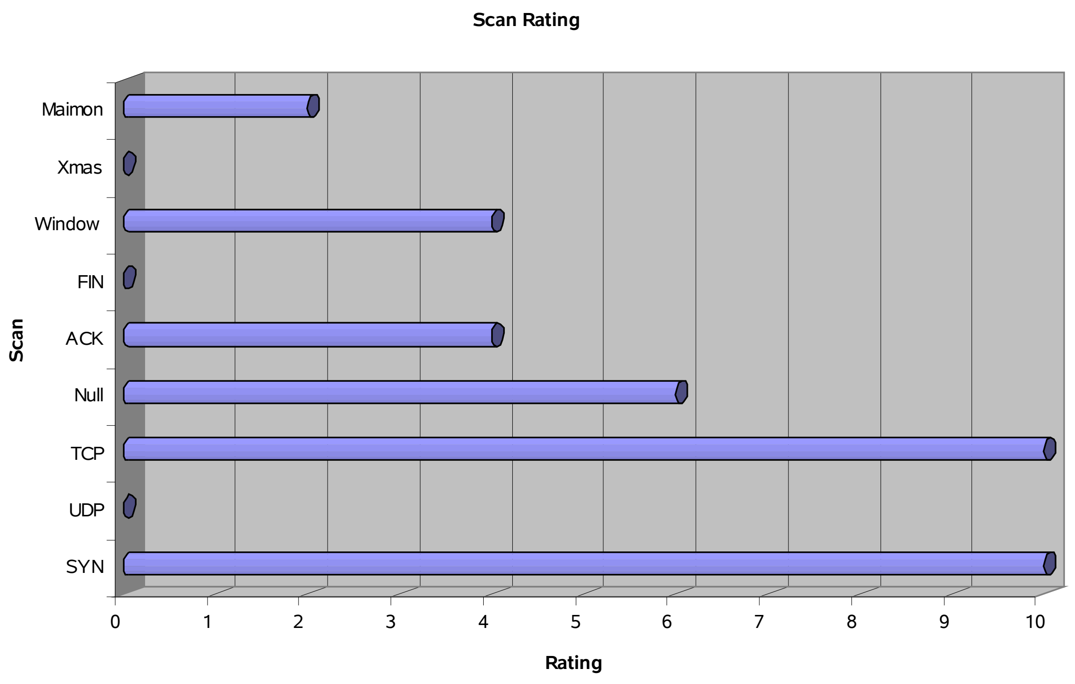


Figure 11 – Comparison of the different scanning ratings and their relative effectiveness

The results that were found regarding the SSH port was expected given the firewalls configurations. This SSH service was the only one enabled on the system, and therefore the other services that were found was not expected.

Fyodor (http://www.insecure.org/nmap/idlescan.html) suggests using countermeasures including a stateful firewall and egress filtering. According to Fyodor, OpenBSD, Solaris and Linux systems are immune to being a zombie system when Nmap performs IPID scans. Linux 2.4.x uses peer-specific IPID values as well as zeros the IPID fields in packets with the Don't Fragment (DF) bit set. OpenBSD randomises the IPID sequence value, but this leads to a problem where the random IPID value cannot be repeated in the packet stream. He also states that it might not stop all IPID related attacks, and that further investigation should be done for other IPID related attacks. Since Linux has its IPID value set to zero, it is safe against any further type of IPID attack.

In the case where the IPID scans were performed the results showed that a system with a kernel version 2.6.9 was not immune to being a zombie system, however it was immune when a kernel version 2.6.16 was used. This is irrespective to whether the firewall was enabled or not.

Xprobe2 does not have any scanning techniques available for IPv6 at this stage. Nmap on the other hand has three scanning techniques for IPv6, but the only useful scan is the TCP scan. There are no other scans available.

2.2.1 Unique Linux Characteristics

Linux reacts differently to other OS when certain packets are received and also sends packets with unique properties, making it possible for fingerprinting in OS detection. The properties that have been found in scans performed as well as those from Ofir Arkin's paper (Arkin 2001b) are stated below.

Linux replies to Echo Request broadcasts as well as Timestamp Requests. It does not respond to Information request ICMPs aimed at the broadcast address, as well as Address Mask Request ICMP messages.

Fyodor states that the IPID value increments with a value of one, but kernel version 2.4.x and above keeps this field zero when the DF bit is set (Fyodor 1998). This is obviously not true, as seen in the IPID scan performed on a Linux system running a kernel version 2.6.9. The DF bit is always set, even if the variable in the ip_no_pmtu_disc file is set.

The TTL values of the Reply ICMP messages are 64 for kernel version 2.0.x, but kernel version 2.2.x and up have a TTL value of 255. Again these facts were found not to be true in the tests performed with Xprobe2, and that the TTL value was 64 for a kernel version 2.6.16. All the Request ICMP messages have a TTL value of 64 though, except for the Destination Unreachable ICMP message, which had a TTL value of 255 (as seen in Appendix H).

Another interesting observation seen in Appendix H is that the sequence number of the TCP packets that were sent, all have a value of zero (relative sequence number). This does not validate as stated in section *1.2.4 Operating System Fingerprinting,* which states that it should be a true random value.

Other characteristics in the TCP/IP stack of the replies with Linux version 2.2.x and 2.4.x are shown in Table 3 (Arkin 2001b).

Table 3 – Characteristic Responses of Linux

	Information Request	***Timestamp Request***	***Address Mask Request***	***Echo Request***
Precedence not = 0	Not Answering	Not = 0x00	Not Answering	Not = 0x00
TOS not = 0	Not Answering	Not = 0x00	Not Answering	Not = 0x00
Unused = 1	Not Answering	0x1	Not Answering	0x1

Another finding about Linux is that when a Timestamp Request ICMP is received and the Code field has a value other than 0, then it will reply with a Timestamp Reply ICMP message, with the Code field value of 0.

Another characteristic of Linux is that it quotes more than 8 data bytes in the ICMP error messages. It will also send ICMP error messages that are up to 576 bytes in size. Twenty bytes are added to the quoting of the corrupt packet with a Destination Unreachable message. This is another characteristic of Linux other than that the Precedence is equal to 0xc0, which only Linux has.

Linux's other feature is that the TCP options are returned in the following order:

<echoed MSS><no op><no op><timestamp><no op><window scale>

The above order can be a fingerprint for these Linux systems.

In the tests that were performed against the Linux systems, it was noted that the Destination Unreachable ICMP messages were blocked and the TCP SYN packets were dropped when the firewall was disabled. The original installed firewall of Suse 10.1 did make a difference in the port scanning and OS detection, but it was not always sufficient.

2.2.2 Preliminary Countermeasures for Linux

The most secure countermeasures for port scanning is to disable the services and ports not in use and to have the latest kernel version installed onto the system, preferably version 2.6.16 or above, as this will prevent the system being a zombie in the IPID scans.

There are many countermeasures suggested, which will be discussed.

Dr. K (Dr. K 2000) suggests that an Intrusion Detection Software (IDS) be used. The best open-source IDS available are SNARE (System iNtrusion Analysis and Reporting Environment) found at www.intersectalliance.com/projects/index.html and SNORT found at www.snort.org. These packages log any scanning and penetration attempts, as well as what might seem to be an attempt. Depending on the configurations, the IDS can notify the system administrator about the attempts, and block any traffic from the specific IP address. SNORT is one of the best IDSs available as it updates its signature database from the Internet when new attacks are found.

Other suggestions are that the system is updated with the latest patches. This will help in the prevention of IP spoofing, as the system could be immune to SYN flooding. Another prevention would be that packets can be dropped that are oversized, as in the case of the Ping of Death. The system administrator can then test the system by running scripts and tools available and then attack it. In this way, the IDS rules and configurations can be altered until the desired results are acquired.

There are times when some services are needed on a system. It is dangerous to have services running that anybody could gain access to. These services should also be made available to those who need them. TCP wrappers gives the possibility of adding Access Control Lists (ACLs) to these services, so that only some people are able to gain access to them (Burgess 2006). With the use of TCP wrappers all communication of the different services are logged. Depending on how secure you need a system to be, one could deny all traffic to it, and then start allowing only systems that require access to it as time progresses. System administrators must investigate the log files for possible attacks.

An internal network can be protected with firewalls. As mentioned previously, stateful firewalls are better. Firewalls can also log half-open connections as well as ICMP messages received.

A stateful firewall with egress filtering is recommended. The firewall should be configured with some of the following rules:

- The firewall itself should not generate TTL Exceeded ICMP messages.

- It should not allow traffic directed to routers, unless it contains routing information.

- Traffic directed to the firewall must be blocked.

- Don't allow ICMP request messages from the Internet to the protected network.

The above countermeasures are preferable for port scanning and further attacks on the system, but countermeasures are needed for OS fingerprinting. Nostromo (Nostromo 2005) suggested patches and programs to fool OS detection scanners such as Nmap and Xprobe2. Some of the suggestions are:

IP Personality by Gaël Roualland et. al. It modifies the TCP ISN, the TCP initial window size, TCP options and IPID numbers. This patch is outdated.

Stealth Patch by Sean Trifero et. al. This patch worked with kernel versions 2.2.19 to 2.2.22 and version 2.4.19. It discards all packets that have the FIN and SYN flags set.

Nostromo (Nostromo 2005) suggests a few other Linux programs, but these are either outdated or do not deceive programs such as Nmap. The above patches change the TCP/IP stack properties to that of other operating systems, but if ICMP scans are performed, then this will have no effect on fooling the attacker.

Morph by Syn Ack Labs (www.synacklabs.net) (Wang 2004) is another OS fingerprint deceiver. Morph alters the TCP/IP packets to have the characteristics of another OS. It deceives QueSO, Nmap and Xprobe2, and it is being developed to deceive P0f and RING/Snacktime.

Another interesting characteristic of Linux is the way it responds to incoming traffic directed to port 0, as described by Ste Jones (Jones 2003). The following seven tests are performed by OS fingerprinting tools that send packets to and from port 0:

- A TCP packet is sent from port 0 to port 0.
- A TCP packet is sent from any port except 0 to port 0.
- A TCP packet is sent from port 0 to an open port.
- A TCP packet is sent from port 0 to a closed port.
- A UDP packet is sent from port 0 to port 0.
- A UDP packet is sent from port 53 to port 0.
- A UDP packet is sent from port 0 to a closed port.

Different OSs reply in different ways to the above tests. The tools that is used to perform these scans is gobbler-2.0.1-alpha (www.networkpenetration.com). Linux replies to all the tests performed. The best countermeasure for this is to block all traffic sent to port 0.

Spangler (Spangler 2003) and Beardsley (Beardsley 2003) suggest that countermeasures for the Ring / Snacktime fingerprinting are that a firewall with filtering is used and that the unused ports are kept closed. A packet mangler, which alters the fields in a packet header, should be used to give an inaccurate retransmission timeout values. The number of retries should also be altered as well as the delays between them. Ofir Arkin (Arkin

2001b) suggests the following countermeasures to prevent scans that use ICMP messages:

- All ICMP Echo messages must be blocked, and hosts on the protected networks must be configured so that they will not answer to ICMP Echo Request messages. This includes ICMP Echo request messages that are sent to the Broadcast Address of the connected network. Should any Echo ICMP messages be needed to be sent then all Echo ICMP messages that contain data must be blocked. This will prevent back doors for programs such as LOKI.

- Block Timestamp Request, Information Request and Address Mask Request ICMP messages and configure hosts to ignore them. This will also prevent Non-Echo Sweeps.

- IP directed broadcasts must be blocked as well, as this will prevent Non-Echo ICMP Broadcasts.

- All fields in the ICMP messages must be checked by the firewall, to prevent Parameter Problem ICMP messages.

- Block Destination Unreachable ICMP error messages from the protected network to the Internet.

- Packets that contain protocols that are not supported must be blocked.

- All outgoing Fragment Reassembly Time Exceeded ICMP error messages must be blocked.

It is concluded that there are countermeasures to port scanning. It is possible to keep a network safe when certain traffic is blocked. Yet there is still a vulnerability hole in terms of OS fingerprinting. Even though there might be patches available, they are outdated and not effective.

IPv6 is currently more secure in terms of port scanning techniques, as there is only a TCP scan available with Nmap 4.00, and Xprobe2 does not support it. The possibility of other scanning techniques using IPv6 is a topic that needs further investigation.

Nmap 4.00 also does not have the possibility of OS fingerprinting for IPv6, which makes a system secure from this point of view. OS detection with IPv6 is also a topic that needs further investigation.

Preventions to IPv6 port scanning will be developed as the new scanning techniques are found. People around the world currently do not use IPv6 unless they are forced to. As it could be years until the Internet works entirely only on IPv6, users will continue to use IPv4. As a result, people will have to think of countermeasures to protect their systems in IPv4 from being scanned. As stated before, there are many countermeasures for port scanning, but countermeasures for OS detection are still lacking.

2.2.3 Summary

The observations found from the port scanning experiments were discussed. The candidate properties that Linux has in it's TCP/IP packets were discussed. Comments were made on the recommended and previous countermeasures that were made available. Other recommendations were also discussed.

2.3. Validation of Preliminary Countermeasures

Programs such as IP Personality fool Nmap and Xprobe2 by changing the TCP packet fields such as the Initial Window Size. These values could be initially set by developers for optimization or sane defaults. It is therefore suggested that these fields and properties are left as the default value and changes are specifically made that have no effect on the TCP/IP stack except those fields that are unique to Linux and have no effect on communication. The order of impact that the properties of Linux have on OS fingerprinting are discussed below, with their respective solution.

Have a firewall implemented. This is probably the most important factor. This will prevent packets with an unknown purpose being sent to your system. As stated before, a stateful firewall is recommended.

The factor for fingerprinting Linux and which carries the largest weighting, is the Precedence value of the TOS field. If this value could be 0x00, then this will be a start in preventing Nmap and Xprobe2 from detecting that Linux is the OS. Only IPv4 has a Type of Service field, so IPv6 is immune to this. It therefore does not affect communication over a network, and it can be concluded that it is safe to change this field.

It was suggested previously that all ICMP messages should be blocked. This is not always needed, as it depends on how secure the requirements for the system are. The characteristic of Linux is that when it sends ICMP error messages more than 8 bytes of the data are quoted, and an extra 20 bytes are added. If only 8 bytes are quoted as done in other operating systems, and the extra arbitrary 20 bytes are removed, then it will also help with the preventing of the OS fingerprinting. This can be achieved by altering the code for the TCP/IP stack.

Some OS fingerprinting programs look at the delays between the ICMP reply messages. To prevent this, the ICMP reply messages can have random delays between them, but this could have a low priority for a countermeasure.

It is also recommended that the SYN/ACK retries are altered to prevent Ring/Snacktime from fingerprinting the OS. This can be done by altering the /proc/sys/net/ipv4/tcp_synack_retries file. The default value is 5, but this value should not be higher than 255. This won't alter the delays between the retries.

The TCP options are in the format:

<echoed MSS><no op><no op><timestamp><no op><window scale>

The order of the options should not have an effect on the TCP/IP communication. Altering the order of the options will therefore have no effect on Linux. Pirating the options order of another OS could work as a countermeasure, but this has a very low impact on the current fingerprinting techniques.

Another noticeable characteristic of Linux which is only used to differentiate between the versions of the kernel is the TTL value. Even though the value of 64 dominates, there are cases where 255 is used. The value of 255 might be used for the TTL field as an optimization value as it was experienced that packets were lost, but the value of this field has a very low impact on the fingerprinting process.

As seen in the tests performed (Appendix H), Linux has a window size of 6840, except for the RST/ACK packets which has a window size of 0. Altering this value will change the performance of the communication across the network. It therefore has a low importance value for a countermeasure. If OS fingerprinting programs depended purely on whether the window size is equal to 6840, then the altering of the default value with a value of one will be enough for a countermeasure, which is not the case with Nmap.

The OS fingerprinting done through the connection of port 0 is not really a threat for Linux at this stage, as Microsoft Windows is an OS that also replied to all seven tests. Even though this might be the case, it is recommended that all traffic to port 0 must be blocked, as this fingerprinting in conjunction with another one could identify the OS. A typical IPtables configuration is as stated below:

```
iptables -I INPUT -p tcp --dport 0 -j DROP

iptables -I INPUT -p udp --dport 0 -j DROP

iptables -I INPUT -p tcp --sport 0 -j DROP

iptables -I INPUT -p udp --sport 0 -j DROP
```

From the above stated countermeasures, it was believed by just having the precedence bits changed to 0x00 would be sufficient to prevent the fingerprinting of Linux. Other fields in the TCP/IP stack can be changed, but these either won't have much influence or they could hamper communication. Further investigation can be done once the change in the precedence bits has been implemented and the fingerprinting is successful.

2.3.1 Type Of Service

Testing to determine whether the TOS has an effect on the OS fingerprinting needs to be performed. Linux default value of the TOS field is 0xc0. IPtables are used to alter the outgoing packets. The code is part of the script found in Appendix B.3. Test code for the IPtables is needed and the following code alters the TOS field:

```
iptables -t mangle -A PREROUTING -j TOS --set-tos 0x00
```

By using the above code the “mangle” table is altered. The TOS field is set to 0x00. The rules for the firewall can be seen in Appendix B.3. Tests were performed by Nmap and Xprobe2 to determine the effect that this change has had on the fingerprinting.

Two systems were used for the tests performed. Both systems were running SUSE 10.1 with the Linux kernel version 2.6.16. The attacking system's IP address was 10.0.0.5 and the target's IP address was 10.0.0.6 and it had port 22 (SSH) open.

The firewall is enabled in all these tests, as IPtables sets the rules for the firewall.

2.3.1.1 Nmap's Results

When Nmap was run with the TOS field set to 0x00, the following results were obtained:

```
linux:/# nmap -O -vv 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-04 02:57 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 02:57
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.01s. Mode: Async [#: 2, OK: 1, NX: 0, DR: 0, SF: 0,
TR: 1, CN: 0]
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) [1672 ports]
at 02:57
Discovered open port 22/tcp on 10.0.0.6
The SYN Stealth Scan took 21.38s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) appears to be up ... good.
Interesting ports on pc-wvl-6.cs.ukzn.ac.za (10.0.0.6):
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0F:FE:31:D9:CF (G-pro Computer)
Device type: general purpose|broadband router
Running: Linux 2.4.X|2.5.X|2.6.X, D-Link embedded
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.4.20, Linux 2.4.26, Linux 2.4.27 or
D-Link DSL-500T (running linux 2.4), Linux 2.4.7 - 2.6.11, Linux 2.6.0 - 2.6.11
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=33DCCB%IPID=Z)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=3398859 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 23.771 seconds
               Raw packets sent: 3364 (135KB) | Rcvd: 20 (1056B)
```

The observation from the above code is that Nmap was not deceived. Tcpdump was executed at the same time, and the following output was received from it:

```
pc-wvl-6.cs.ukzn.ac.za.ssh > pc-wvl-5.cs.ukzn.ac.za.51941: S, cksum 0x66af
(correct), 3424862189:3424862189(0) ack 561094594 win 5792
<mss1460,nop,nop,timestamp 129682542 1061109567,nop,wscale 2>
02:58:00.013585 IP (tos 0x0, ttl  64, id 0, offset 0, flags [DF], proto: TCP (6),
length: 40)
```

From the Tcpdump output it is evident that the TOS field is 0x00.

2.3.1.2 Xprobe2's Results

The following results were observed from Xprobe2:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.01042 sec
[+] Selected safe Round-Trip Time value is: 0.02083 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1" (Guess probability: 100%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.21" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.22" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.04T53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.05KT53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.01BT51" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.04aT51" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.7.01eT53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.23" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.24" (Guess probability: 91%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

From the output, Xprobe2 guessed that the OS of the target is either Foundary Networks IronWare or Linux.

Changing the TOS field did not prevent the OS fingerprinting tools from detecting the OS running on the target.

Other changes to the TCP/IP packet are needed to deceive the OS fingerprinting tools.

2.3.2 ICMP Echo Ignore All

For the tests that follow, the variables were altered that were in the /proc/sys/net/ipv4/ directory. These are the settings that the Linux kernel is currently working on. To alter these variables root access is needed.

The default setting for this variable is zero. Should this variable be non-zero, then the kernel will ignore all ICMP echo requests. This variable was changed with the following statement:

```
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

2.3.2.1 Nmap's Results

The following output was observed from Nmap:

```
linux:/# nmap -O -vv 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-04 03:22 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 03:22
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.04s. Mode: Async [#: 2, OK: 1, NX: 0, DR: 0, SF: 0,
TR: 1, CN: 0]
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) [1672 ports]
at 03:22
Discovered open port 22/tcp on 10.0.0.6
```

```
SYN Stealth Scan Timing: About 32.08% done; ETC: 03:23 (0:01:03 remaining)
The SYN Stealth Scan took 67.31s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) appears to be up ... good.
Interesting ports on pc-wvl-6.cs.ukzn.ac.za (10.0.0.6):
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0F:FE:31:D9:CF (G-pro Computer)
Device type: general purpose|broadband router
Running: Linux 2.4.X|2.5.X|2.6.X, D-Link embedded
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.4.20, Linux 2.4.26, Linux 2.4.27 or
D-Link DSL-500T (running linux 2.4), Linux 2.4.7 - 2.6.11, Linux 2.6.0 - 2.6.11
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=2C977C%IPID=Z)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=2922364 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 70.019 seconds
               Raw packets sent: 5045 (203KB) | Rcvd: 27 (1350B)
```

From the output it is evident that Nmap was able to determine that the OS running on the target system was Linux.

2.3.2.2 Xprobe2's Results

Xprobe2 was executed and the following was observed:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is down (Guess probability: 0%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Xprobe2 was not able to determine what the OS was as it did not receive any ICMP reply packets. Xprobe2 relies purely on the ICMP packets it receives. This is a disadvantage of Xprobe2. Nmap on the other hand was able to detect the OS.

With the ICMP packets disabled there is no need to alter the packets in terms of the amount of data sent in the ICMP error packet.

2.3.3 IP Default TTL

The impact that the TTL value has on the OS fingerprinting tools has to be investigated. This is done by altering the TTL value in the /proc/sys/net/ipv4/ip_default_ttl file. The default value is 64. It was decided to modify the TTL value to any random number below 256. The TTL value was changed to 128. It was reasoned that it might be a countermeasure to have this value a variable number, but it does bring in a problem that this could be a vulnerability in detecting Linux that has a variable TTL. It would be advisable not to have the TTL value below 64, as this is a reasonable number to prevent a packet from getting lost on the Internet.

2.3.3.1 Nmap's Result

The results that Nmap displayed are shown below:

```
linux:# nmap -O -vv 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-04 03:47 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 03:47
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.01s. Mode: Async [#: 2, OK: 1, NX: 0, DR: 0, SF: 0,
TR: 1, CN: 0]
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) [1672 ports]
at 03:47
Discovered open port 22/tcp on 10.0.0.6
The SYN Stealth Scan took 21.38s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) appears to be up ... good.
Interesting ports on pc-wvl-6.cs.ukzn.ac.za (10.0.0.6):
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0F:FE:31:D9:CF (G-pro Computer)
Device type: general purpose|broadband router
Running: Linux 2.4.X|2.5.X|2.6.X, D-Link embedded
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.4.20, Linux 2.4.26, Linux 2.4.27 or
D-Link DSL-500T (running linux 2.4), Linux 2.4.7 - 2.6.11, Linux 2.6.0 - 2.6.11
OS Fingerprint:
TSeq(Class=RI%gcd=3%SI=A7336%IPID=Z)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=684854 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 23.758 seconds
               Raw packets sent: 3364 (135KB) | Rcvd: 19 (996B)
```

Nmap was not deceived by this alteration.

2.3.3.2 Xprobe2's Results

Xprobe2 was executed to investigate the affect of the alteration on it.

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00026 sec
[+] Selected safe Round-Trip Time value is: 0.00051 sec
```

```
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1"
(Guess probability: 91%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.0" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.1" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.2" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.3" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.4" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.5" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.6" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.7" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.6.8" (Guess probability: 83%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The results show that Xprobe2 was effected in some way. Its primary guess was still Foudary IronWare, but with the other tests it performed it guessed the OS of the target to be Linux, with an 83% probability. Interestingly enough, it was found that when the TTL value was set to 255, Xprobe2 gave a different result, but Nmap was not effected.

The following results were found from Xprobe2.

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
```

```
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00362 sec
[+] Selected safe Round-Trip Time value is: 0.00724 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.5" (Guess probability: 91%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.1" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.5" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.20" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.24" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.16" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "NetBSD 2.0" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.0" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.4" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.2.19" (Guess probability: 91%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

From the results Xprobe2's primary guess was that the target's OS was Linux. The other guesses were that the OS of the target is Linux and in one situation that it was NETBSD 2.0. This shows that the TTL value has some effect on Xprobe2, but it does not fully deceive it.

2.3.4 TCP Window Scaling

TCP window scaling is defined according to RFC 1323. The /proc/sys/net/ipv4/tcp_window_scaling file determines if the kernel follows these definitions or not. The default value is 1 which is to follow these definitions. The value in this file was set to 0.

Investigation was done on how this change affected Nmap and Xprobe2.

2.3.4.1 Nmap's Output

Nmap gave the following output:

```
linux:/# nmap -O -vv 146.2 30.94.108

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-04 04:06 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 04:06
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 0.01s. Mode: Async [#: 2, OK: 1, NX: 0, DR: 0, SF: 0,
TR: 1, CN: 0]
```

```
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) [1672 ports]
at 04:06
Discovered open port 22/tcp on 10.0.0.6
The SYN Stealth Scan took 21.39s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host pc-wvl-6.cs.ukzn.ac.za (10.0.0.6) appears to be up ... good.
Interesting ports on pc-wvl-6.cs.ukzn.ac.za (10.0.0.6):
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0F:FE:31:D9:CF (G-pro Computer)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.6.8 - 2.6.11
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=2888C6%IPID=Z)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNT)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=2656454 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 23.785 seconds
               Raw packets sent: 3364 (135KB) | Rcvd: 18 (904B)
```

This alteration had some effect on Nmap. Nmap was still able to determine that Linux was running on the target system, but with fewer kernel versions. This means that this is not the only thing Nmap looks at to determine the OS.

2.3.4.2 Xprobe2's Results

Xprobe2 gave the following results:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00027 sec
[+] Selected safe Round-Trip Time value is: 0.00054 sec
```

```
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1"
(Guess probability: 91%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.05KT53"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.01BT51" (Guess
probability: 83%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.04aT51" (Guess
probability: 83%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.7.01eT53" (Guess
probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.25" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.24" (Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM H.07.15 EEPROM H.08.20"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.08.21 EEPROM G.08.21"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.08.08 EEPROM G.08.04"
(Guess probability: 83%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

From the output it is seen that Xprobe2 thought the target's OS was between Foundary IronWare, Linux and HP JetDirect ROM. This has had the greatest effect on Xprobe2. It was then decided to have the TTL value altered to 128 while the tcp_window_scaling file is set to 0.

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu
```

```
[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00029 sec
[+] Selected safe Round-Trip Time value is: 0.00059 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1"
(Guess probability: 83%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM A.03.17 EEPROM A.04.09"
(Guess probability: 83%)
```

```
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM A.05.03 EEPROM A.05.05"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM F.08.01 EEPROM F.08.05"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM F.08.08 EEPROM F.08.05"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM F.08.08 EEPROM F.08.20"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.05.34 EEPROM G.05.35"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.06.00 EEPROM G.06.00"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.07.02 EEPROM G.07.17"
(Guess probability: 83%)
[+] Host 10.0.0.6 Running OS: "HP JetDirect ROM G.07.02 EEPROM G.07.20"
(Guess probability: 83%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Here Xprobe2 guessed that the OS of the target was either Foundary IronWare or HP JetDirect ROM, with no mention of Linux while Nmap had no change in its output.

From the above tests it was established that if the Linux kernel was not following RFC 1323, then it was able to deceive Xprobe2 but not Nmap. Causing the kernel not to follow RFC 1323 can result in the communication not being optimised, so it is recommended that this value is kept as it is.

As Xprobe2 was deceived when Nmap was not, it can be said that Nmap is a more powerful tool than Xprobe2 when it comes to OS fingerprinting. It is more difficult to deceive Nmap than Xprobe2.

The reason why Nmap is a better OS fingerprinting tool is due to the fact that it has 1681 different fingerprints of tests performed on OS in its database (nmap-os-fingerprint). This database is community maintained. It makes it very difficult to deceive Nmap, as Fyodor has 172 fingerprints for Linux with different alterations done to the TCP/IP stack. As Xprobe2 is deceived more easily, more focus is needed to investigate countermeasures for Nmap instead of Xprobe2.

2.3.5 Timestamps

All the settings that were changed were reset manually, but the TOS field was kept to 0x00. It was then decided to disable the TCP timestamps in the /proc/sys/net/ipv4/tcp_timestamps file.

With this change in the TCP/IP stack, only Nmap's results were observed.

2.3.5.1 Nmap's Results

Nmap showed the following output:

```
linux:/# nmap -O -vv 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-05 15:45 SAST
Initiating ARP Ping Scan against pc-wvl-6.cs.ukzn.ac.za [1 port] at 15:45
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za [1672 ports] at 15:45
Discovered open port 22/tcp on  pc-wvl-6.cs.ukzn.ac.za
SYN Stealth Scan Timing: About 32.08% done; ETC: 15:46 (0:01:03 remaining)
The SYN Stealth Scan took 67.31s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host pc-wvl-6.cs.ukzn.ac.za appears to be up ... good.
Interesting ports on pc-wvl-6.cs.ukzn.ac.za:
```

```
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:30:18:66:7A:0B (Jetway Information Co.)
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux 2.4.0 - 2.5.20 w/o tcp_timestamps, Linux 2.4.22 (x86)
w/grsecurity patch and with timestamps disabled
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=3C3087%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=MNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16D0%ACK=S++%Flags=AS%Ops=MNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=N)
PU(Resp=N)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=3944583 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 82.684 seconds
               Raw packets sent: 5045 (203KB) | Rcvd: 27 (1254B)
```

From these results, it can be seen that Nmap still detects Linux, but thinks that a patch from grsecurity has been installed. This is an example of the vast number of fingerprints that Nmap has. It is observed that the window size has changed to 16D0. This suggests that the window size be changed to another value to deceive Nmap.

2.3.6 Other modifications of the TCP/IP stack

Modifying the Window size and MSS field of the TCP/IP stack were considered, but this would be ineffective, as Fyodor could just add these fingerprints to the Nmap database, unless the alterations mimic another OS. No matter what changes are made to the TCP/IP stack, it would have no effect on Nmap, except if one would want to make the TCP/IP stack the same as that of another OS. Modifying the TCP/IP stack to be the same as that of another OS creates the problem that the system could have vulnerabilities or problems equivalent to that of the other OS. An example is that if the IPID bit is not set to zero, and the DF bit is not set, then the system could be vulnerable to IPID scans.

It can therefore be concluded that the modifications of the TCP/IP stack will be useful to a degree, but it would not be the best countermeasure for OS fingerprinting.

It is suggested that the ICMP packets discussed in section *2.2.1 Unique Linux Characteristics* are blocked as this will prevent OS fingerprinting through a firewall or a system. Tools that operate similarly to Xprobe2 and that are reliable on ICMP messages will not be able to fingerprint the OS of a target system successfully.

Chapter 3: Countermeasures

Based on the results of tests to validate the preliminary countermeasures in Section 2.3, it was decided that another approach should be investigated and tested in addition – the active detection of OS fingerprinting tools. The main difference between this technique and the others is that if the target can detect an OS fingerprinting attempt, it could respond in a more intelligent way while maintaining 'normal' networking functionality for other communication.

Once this approach has been investigated, the final set of countermeasures is presented. This includes altering the TCP/IP stack, as well as the afore-mentioned approach of detecting scans performed on a system. This final list of countermeasures make a system as secure as possible against OS fingerprinting.

3.1 OS Fingerprinting Tools Detection

According to McClure et. al. (McClure *et al.* 2005), OS fingerprinting can only be prevented by modifying the unique TCP/IP stack fingerprint, but this will "*affect the functionality of the operating system.*" As this is not the desired option, the task is therefore to try and prevent a scan from Nmap, without modifying the TCP/IP stack.

3.1.1 Detecting Nmap

The database of P0f (Zalewski 2004) was investigated, and it was noted that Nmap itself has a fingerprint because of the unique way it sends the probing packets to a target system. It was then thought to use the same tool against the attacker that is used on the target. By this it is meant that one should monitor the packets being received and identify if they come from Nmap. Should the packets come from Nmap then the system should either drop them or respond appropriately.

As Nmap is not able to determine the OS if it cannot find at least one open port, a system could react to all probes from Nmap as if the port is closed. Evgeniy Polyakov used the fact that Nmap has a unique signature and combined this with IPtables, to design the OSF (Operating System Fingerprinting) module. OSF can be downloaded from http://tservice.net.ru/~s0mbre/archive/osf/. The OS fingerprints can be downloaded from http://www.openbsd.org/cgi-bin/cvsweb/src/etc/pf.os. Evgeniy Polyakov states that this is the best countermeasure for active OS fingerprinting tools such as Nmap. This won't prevent passive OS fingerprinting tools, but a user from the target system would have to connect to the attacker's machine first. The task is rather to block active OS fingerprinting tools.

The procedure to get OSF working is really involved and somewhat complicated for first time users. The Makefile is first edited and the path of the IPtables source files is specified. Once this is done, the source code is built that generates the libipt_osf.ko kernel module. The library file is then also compiled which generates the libipt_osf.so shared library. The kernel module is then installed and the pf.os file is loaded into the /proc/sys/net/osf path. Once this has been done, it can be implemented with Iptables. The installation procedure is described in Appendix D.

Nmap has the fingerprints presented below. The order of the fingerprints are as follows:

<Window Size>:<Initial TTL>:<Don't Fragment bit>:<Overall SYN packet size>: <Options in Order if used>: Nmap scan or OS

1024:64:0:40:	Nmap SYN scan 1
2048:64:0:40:	Nmap SYN scan 2
3072:64:0:40:	Nmap SYN scan 3
4096:64:0:40:	Nmap SYN scan 4
1024:64:0:60:W10,N,M265,T:	Nmap OS detection probe 1
2048:64:0:60:W10,N,M265,T:	Nmap OS detection probe 2
3072:64:0:60:W10,N,M265,T:	Nmap OS detection probe 3
4096:64:0:60:W10,N,M265,T:	Nmap OS detection probe 4

NAST, another OS fingerprinting tool has got the following fingerprint:

32767:64:0:40:	NAST SYN scan

Due to the fact that Windows 2003 has a similar fingerprint to Nmap, it is suggested that the statement for the detecting of Nmap is placed at the end of the IPtables list, therefore allowing communication from Windows 2003. The main options used are --log and --ttl.

The statement used for the blocking of Nmap packets is:

```
iptables -I INPUT -p tcp -m osf --genre Nmap --log 2 --ttl 2 -j REJECT
```

The above statement rejects all TCP packets coming in from Nmap.

The firewall settings are shown in Appendix B.3, which were used once the OSF module was loaded. When Nmap was used to fingerprint the OS of the target system, the following output was observed:

```
linux:/# nmap -O -vv  10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-09-15 23:48 SAST
Initiating ARP Ping Scan against pc-wvl-6.cs.ukzn.ac.za [1 port] at 23:48
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0, TR: 3, CN: 0]
Initiating SYN Stealth Scan against pc-wvl-6.cs.ukzn.ac.za [1672 ports] at 23:48
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 0 to 5 due to max_successful_tryno increase to 4
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 5 to 10 due to max_successful_tryno increase to 5
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 10 to 20 due to max_successful_tryno increase to 6
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 20 to 40 due to max_successful_tryno increase to 7
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 40 to 80 due to max_successful_tryno increase to 8
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 80 to 160 due to max_successful_tryno increase to 9
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 160 to 320 due to 11 out of 12 dropped probes since last increase.
SYN Stealth Scan Timing: About 2.82% done; ETC: 00:06 (0:17:22 remaining)
```

```
Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 320 to 640 due to 11 out of
11 dropped probes since last increase.

Increasing send delay for pc-wvl-6.cs.ukzn.ac.za from 640 to 1000 due to 11 out of
19 dropped probes since last increase.

SYN Stealth Scan Timing: About 65.20% done; ETC: 00:15 (0:09:32 remaining)

The SYN Stealth Scan took 1668.84s to scan 1672 total ports.

Warning:  OS detection will be MUCH less reliable because we did not find at least
1 open and 1 closed TCP port

Host pc-wvl-6.cs.ukzn.ac.za appears to be up ... good.

All 1672 scanned ports on pc-wvl-6.cs.ukzn.ac.za are: filtered

MAC Address: 00:30:18:66:7A:0B (Jetway Information Co.)

Too many fingerprints match this host to give specific OS details

TCP/IP fingerprint:

SInfo(V=4.00%P=i586-suse-linux%D=9/16%Tm=450B2637%O=-1%C=-
1%M=003018)

T5(Resp=N)

T6(Resp=N)

T7(Resp=N)

PU(Resp=N)


Nmap finished: 1 IP address (1 host up) scanned in 1690.096 seconds

            Raw packets sent: 1851 (76.1KB) | Rcvd: 1673 (114KB)
```

As it can be seen from the above output, Nmap was not able to determine the OS running on the target system as it was not able to find an open port.

The file /var/log/messages was investigated, and the following was seen:

```
Sep 15 11:48:13 linux-xegw kernel: ipt_osf: Windows [.NET::Windows .NET Enterprise Server] : 10.0.0.5:57418 -> 10.0.0.6:716 hops=206

Sep 15 11:48:13 linux-xegw kernel: ipt_osf: NMAP [syn scan:2:NMAP syn scan (2)] : 10.0.0.5:57418 -> 10.0.0.6:716 hops=15
```

OSF first thought that it was receiving packets from a Windows .NET Enterprise Server, and then it found that it was receiving packets from Nmap. With the '-j REJECT' option, the system will send ICMP error messages to the attacking system, if ICMP messages are not disabled. It is therefore better to have the line:

```
iptables -I INPUT -p tcp -m osf --genre Nmap --log 2 --ttl 2 -j DROP
```

Vmstat was executed when the OSF module was installed and removed, and the following was shown respectively:

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 2  0      0 400708  47268 451152    0    0   122    31  279   320  8  1 88  3
```

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
 1  0      0 421140  43220 436220    0    0   148    34  278   351 10  1 86  3
```

From this it is seen that with the OSF module installed, less free memory was available, but this could also be due to other programs that are running. The time that was spent running kernel code (in system time) does not differ.

It can be concluded that OSF solves the problem of active OS fingerprinting by Nmap as well as port scanning, and is therefore the best countermeasure found. As the target system logs the IP address of the system performing the scans, the system administrator can block any further packets received from that system.

3.1.2 Detecting Other OS Fingerprinting Tools

Investigation was done to see how other OS fingerprinting tools reacted to this modification on the target system. A set of OS fingerprinting tools was tested to determine performance against the target.

Xprobe2 was tested with the OSF module enabled on the target system, and the following output was received:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00024 sec
[+] Selected safe Round-Trip Time value is: 0.00049 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip
for www.securityfocus.com in UDP probe
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1"
(Guess probability: 100%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.21" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.22" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.04T53"
(Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.05KT53"
(Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.01BT51" (Guess
probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.04aT51" (Guess
probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.7.01eT53" (Guess
probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.23" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.24" (Guess probability: 91%)
```

From the above results it is seen that Xprobe2 was not effected by the installation of the OSF module on the target system. This was expected as Xprobe2 mainly gets its results from returned ICMP messages.

QueSo was tested and it gave the following output:

```
linux:/# queso 10.0.0.6:22
10.0.0.6:22   * Standard Solaris 2.x, Linux 2.2.??? 2.4.???, MacOS
```

This output appeared even if OSF was not loaded. QueSo was not able to determine the exact OS running on the target system.

It was then decided that SAINT (Security Administrator's Integrated Network Tool) must be used to determine how secure the target is (found at www.wwdsi.com/saint). The following output was seen when OSF was not loaded:

```
linux:/# saint 10.0.0.6
> bin/udp_scan: are we talking to a dead host or network?
>
10.0.0.6:
   Services:
      auth
      SSH
```

The OSF module was then loaded, and the following output was observed from SAINT:

```
linux:/# saint 10.0.0.6
> > bin/udp_scan: are we talking to a dead host or network?
```

As seen above, when OSF was loaded, SAINT was not able to detect even open ports from the target.

Strobe (http://linux.maruhn.com/sec/strobe.html) was also tested. Strobe was not able to detect any open ports on the target system when a general port scanning test was performed. The target system's /var/log/messages file gave the following:

```
Jul 13 11:53:02 linux-g5ii sshd[12819]: Did not receive identification string from 10.0.0.5
```

The following output was given the Strobe did a scan specifically on port 22 (SSH).

```
linux:/# strobe 10.0.0.6:22
strobe 1.05 © 1995 – 1999 Julian Assange <proff@iq.org>
attempting          port=22             host=10.0.0.6
10.0.0.6    22 ssh                 #SSH Remote Login Protocol
                                   -> SSH-1.99-OpenSSH-4.2\n
```

The file /var/log/messages showed the following:

```
Jul 13 11:48:07 linux-g5ii sshd[11989]: Did not receive identification string from 10.0.0.5
```

This proves that strobe is not as good as Nmap in doing general port scans and open ports are not known. Strobe was able to identify that port 22 was open and the service that was running on it.

McClure *et. al.* highly recommends a Microsoft Windows based port scanner and OS fingerprinter, NetworkActiv Port Scanner. This OS fingerprinting tool uses the same principle as Xprobe2, in that ICMP packets are sent to the target. With the target allowing ICMP messages, NetworkActiv Port Scanner reported the following OS running on the target:

```
Primary guess(es) with 100% Match:

 MacOSX

 Linux kernel 2.0.29

 Linux kernel 2.2.10

 Linux kernel 2.2.14-20000612

 Linux kernel 2.2.16C32III

 Linux kernel 2.2.19-3cl

 Linux kernel 2.2.20

 Linux kernel 2.4.2-2

 Linux kernel 2.4.7-10

 Linux kernel 2.4.9-6

 Linux kernel 2.4.18

 Linux kernel 2.4.18-3

 GNU/Linux 2.1

 GNU/Linux 3.0

 SunOS 5.6

 SunOS 5.8

 Solaris 8

 pSOSystem

 AIX

 FreeBSD

 FreeBSD/i386

 HP JetDirect


Secondary guess(es) with 0% Match:

Unknown
```

All ICMP messages were then rejected, and NetworkActiv Port Scanner reported the following OS on the target:

```
Primary guess(es) with 100% Match:
 MacOSX
 Linux kernel 2.0.29
 Linux kernel 2.2.10
 Linux kernel 2.2.14-20000612
 Linux kernel 2.2.16C32III
 Linux kernel 2.2.19-3cl
 Linux kernel 2.2.20
 Linux kernel 2.4.7-10
 Linux kernel 2.4.9-6
 Linux kernel 2.4.18
 Linux kernel 2.4.18-3
 GNU/Linux 2.1
 GNU/Linux 3.0
 Solaris 8
 FreeBSD
 FreeBSD/i386
 HP JetDirect

Secondary guess(es) with 66% Match:
 Windows ME - on Ethernet
 Windows 2000 Professional - Stock/SP1/SP2
 Windows 2000 Professional - Stock/SP1/SP2 on Ethernet
 Windows 2000 Professional - SP3 on Ethernet
 Windows 2000 Professional - SP3
 Windows XP Home Edition - on Ethernet
 Windows XP Professional - on Ethernet
 Windows XP Professional
 Linux kernel 2.4.2-2
```

Solaris

pSOSystem

Netopia R5200-K v4.3.8

Cisco 3620 WAN Router

Cisco 6509/7200 Router

Cisco GSR 12016

The result of the above output indicates that NetworkActiv Port Scanner is not as specific as Nmap, and was not able to pin-point the OS running on the target.

This again indicates that Nmap is a better OS fingerprinting tool. With the above tests performed, it shows that OSF block packets from Nmap but not other OS scanning tools. OSF also has the advantage that should a new type of OS fingerprinting tool become available, its fingerprint can be added to OSF's database.

The attacker might be able to realise that he is being countered, when he is trying to scan ports of a system with Nmap and the results show that the ports are all closed, especially, for example, when the attacker is able to connect to a web server at the same IP address with a web browser. This is a disadvantage of the current version of the OSF module, and this will result in a cycle of improvements that will be made by the developers of the OSF module and Nmap.

The only way Nmap can do an OS fingerprint of a system, is when the data length of the packet is changed. This is a weakness of the OSF module. To counteract this vulnerability, the OSF module should increase the TCP/IP analysis of more fields compared to the few fields currently observed.

3.2 Final Countermeasures

Looking at the overview of all the preliminary countermeasures discussed, it is suggested that a stateful firewall be used to prevent OS fingerprinting. In Linux, this is possible by using IPtables. The different features of the required firewall configuration are discussed, as well as how to implement this configuration with the use of IPtables.

Different systems will have different firewall configurations of the settings needed, depending on what the system is to be used for. For example. a system could be a PC, firewall, router, gateway, back-end server, or a combination of one or more of the above. The IPtables rules should then be captured in a parametrised shell script that is executed when the system boots and every time the system takes on a different role.

3.2.1 Type of Service Field

This is one of the major fields that will help prevent OS fingerprinting, not only by active, but also by passive OS fingerprinting tools. Nmap and Xprobe2 first look at this field to determine if the OS is Linux or not. To modify the default value of 0xC0 to 0x00, as other operating systems have it, the following IPtables statement can be used:

```
iptables -t mangle -A PREROUTING -j TOS --set-tos 0x00
```

The above statement could have the '-A POSTROUTING' option as well, but is mainly used for routers. This statement also works for standalone PCs, but the following statement will also work:

```
iptables -t mangle -I OUTPUT -j TOS --set-tos 0x00
```

This statement modifies the packet's TOS field as it is leaving the system. There are different variants of the above statement that can be used depending on the use of the system.

3.2.2 OS Fingerprinting Tools Detection

This is achieved with the use of the OSF module. The installation and loading of this module are performed with the steps shown in Appendix D.

The Nmap packets are blocked with the statement:

```
iptables -I INPUT -p tcp -m osf --genre Nmap --log 2  --ttl 2 -j DROP
```

In case a new OS fingerprinting tool becomes available and OSF is not able to drop the packets from this tool, then the new tools fingerprint can be included in the OSF database. To do this, edit the pf.os file and add the new fingerprint in the format:

```
<Window Size>:<Initial TTL>:<Don't Fragment bit>:<Overall SYN packet size>:      <Options in Order if used>:          OS Fingerprinting Tool's Name
```

It is advised that the IPtables statement is placed at the end of the table, to allow communication with Microsoft Windows system, or any other OS that might have a similar fingerprint as that of the OS fingerprinting tool.

The system administrator can therefore view the /var/log/messages file, and monitor any scans performed against the system. The IP addresses are also logged so all communication from these IP addresses can be blocked.

3.2.3 Port 0 Disabled

All packets that are received that have a source port or a destination port of zero, must be dropped. This will prevent OS fingerprinting from tools that monitor how a system reacts when these ports are used.

The best option is to drop all TCP and UDP packets that are received with these conditions with the use of IPtables. The IPtables statements that are suggested are below:

```
iptables -I INPUT -p tcp --dport 0 -j DROP

iptables -I INPUT -p udp --dport 0 -j DROP

iptables -I INPUT -p tcp --sport 0 -j DROP

iptables -I INPUT -p udp --sport 0 -j DROP
```

3.2.4 Block ICMP messages

As many OS fingerprinting tools use ICMP messages to bypass the firewall and to analyse a system, it is suggested that all replies and broadcasting ICMP messages that are going out, are to be blocked. ICMP messages that are coming in might be helpful to establish if an error has occurred on the network.

This can be done by placing these rules in the /etc/sysctrl.conf file. The following lines must be inserted into this file which will then set the settings for the kernel.

```
net.ipv4.icmp_echo_ignore_all = 1

net.ipv4.icmp_echo_ignore_broadcasts = 1

net.ipv4.icmp_ignore_bogus_error_responses = 1
```

The above statements drop all ICMP echo, broadcast and bogus error messages. They assist in preventing information being sent in ICMP messages that will allow OS fingerprinting.

Due to the fact that the necessary ICMP messages must be blocked, it is therefore suggested that the '-j DROP' option is used instead of the '-j REJECT' option, as these usually send ICMP error messages. This will also prevent Traceroutes of a system.

The blocking of needed ICMP messages can also be done with the use of IPtables. The following line will drop respectively all Echo Reply, Timestamp Reply and Information Reply ICMP messages that are going out.

```
iptables -p icmp --icmp-type 0 -I OUTPUT -j DROP

iptables -p icmp --icmp-type 14 -I OUTPUT -j DROP

iptables -p icmp --icmp-type 16 -I OUTPUT -j DROP
```

This line can be inserted into the same script that the other IPtables statements are in. This statement is probably the most effective, as it won't allow ICMP Reply messages out of the system, but it would allow Request

ICMP message to be sent.

3.2.5 Conclusion

The most effective ways to prevent OS fingerprinting on a Linux system have just been discussed. Recall that they are:

- Modification of the Type of Service field.
- The installation of the OSF module.
- Disabling port 0 communication.
- Blocking certain ICMP messages.

We will now conclude the findings from this project.

Recall that the first contact with a target system is through port scanning. This activity involves searching for open ports as well as determining the operating system running on the target. The supplying of inaccurate information at this stage of the attack, could be useful to prevent the target system from being compromised any further.

IPv4 and OS fingerprinting were investigated and tests were performed to determine how secure the default installation of Suse 10.1 Linux operating system can be configured. It was found that the default installation of Suse 10.1 Linux was not that secure in terms of port scanning and active OS fingerprinting.

IPv6 was investigated and it was found that this is a solution, as Nmap does not support OS fingerprinting in IPv6 at this stage. Since IPv6 is not going to be used worldwide for some time, it was then decided to investigate OS fingerprinting in IPv4.

It was initially thought that modifying the TCP/IP stack will prevent active OS fingerprinting after the techniques for OS fingerprinting were investigated. A number of OS fingerprinting tools were investigated, such as Nmap, Xprobe2, Strobe, SAINT, QueSo and NetworkActive Port Scanner and it was concluded that the best tools available are Nmap and Xprobe2. Xprobe2 was relatively easy to deceive, and it relied on the ICMP messages it received from the target system. Nmap on the other hand was not that easy to deceive, due to the fact that it has such a large database of fingerprints, and should an alteration be made to the TCP/IP stack, this modified fingerprint could simply be added to Nmap's database. The fields in the TCP/IP stack that was modified to try and deceive Nmap were the TOS field, the IP TTL value, the TCP Window Scaling and the timestamps.

Other techniques to deceive OS fingerprinting tools, especially Nmap, were investigated. It was then decided that the technique that is being used against the target system is also to be used against the attacker. This is done by using the fingerprint of Nmap to identify any packets coming from it. All packets that are received from Nmap are simply dropped. This does not only prevent OS fingerprinting against the system but against port scanning as well.

This countermeasure is taken to be the best because when any new types of OS fingerprinting tools become available, it's fingerprint then can simply be added to the database. There were some OS fingerprinting and port scanning tool such as QueSo that was able to perform a fingerprint of the target's OS.

In the cases that Nmap performed a scan against a target system, it was noted that it could not pin-point the OS running on the target. The attacker should be able to identify that the target system has detected a scan that is being performed against it and is blocking the scan. It should also be obvious to the attacker that a module such as OSF is present when it is known that a system has an open port, and a scan cannot be performed. It was found that with the installation of the OSF module, the CPU was not affected with processing overhead.

Nmap can, however, do an OS fingerprint of a system when the data length of the packet is changed. This is a weakness of the OSF module. To counteract this vulnerability, the OSF module should increase the TCP/IP analysis of more fields compared to the few fields currently observed.

An attacker will first try to perform a normal Nmap scan, and only when s/he identifies that a module such as OSF is blocking the packets, will the data length of the scanning packets be changed. The OSF module can be extended so that when it detects an Nmap scan, it replies with a fingerprint of either another OS or another network component, such as a printer or a router. It can be suggested that workstations return a printer's fingerprint, the attacker might become suspicious if a public web server returns a fingerprint of a printer, and will be able to know that the attacks are being counteracted. Therefore it is suggested that servers connected to the Internet return the fingerprint of another OS. There are programs and modules that could do this, but this could be added to the OSF module.

Other countermeasures were also investigated, and it was found that the TOS field in the Linux TCP stack should be set to 0x00 instead of the default 0xC0, which is mostly unique to Linux. Other suggestions are that all packets with a source or destination port of zero be dropped, and that certain ICMP messages leaving the system be blocked. The firewall to the network should also monitor the packets coming in and going out of the network. Packets must have legitimate fields and the source and destination addresses must be valid for the respective network.

With the above suggested countermeasures, it can be concluded that if they are implemented on a system, it will make a system considerably more secure against active OS fingerprinting using currently known techniques.

3.2.6 Future Work

Nmap could be improved by sending a fingerprint of another OS to the target system. The OSF module will then recognise that a system running a legitimate OS, is trying to connect to it and probably accept the connection if these settings are configured in the firewall rules.

A cycle will occur, in that the OSF module will be improved to counteract Nmap, while Nmap will be improved to bypass the OSF module.

An investigation of OS fingerprinting of IPv6 packets will also determine if it has a weakness, resulting that patches and other countermeasures will be needed to be developed.

The OSF module can be extended to have a database that is community maintained as Nmap's database is. For a better detection of fingerprinting tools, the OSF module needs to monitor more fields in the TCP/IP stack, so that fingerprinting tools will not be able to deceive it. An added feature to the OSF module would be that should it detect an OS fingerprinting tool performing a scan against the system, it will reply with another OS's fingerprint.

References

Arkin, O. (2001a) X remote ICMP based OS fingerprinting techniques.

http://www.sys-security.com/archive/papers/X_v1.0.pdf - 1 August 2006.

Arkin, O. (2001b) ICMP Usage in Scanning, Version 3.0.

http://www.blackhat.com/presentations/bh-europe-01/arkin/bh-europe-01-arkin.ppt – 1 August 2006.

Arkin, O., Yarochkin, F. & Kydyraliev, M. (2003) The Present and Future of Xprobe2, The Next Generation of Active Operating System Fingerprinting.

http://sys-security.com/blog/archive/papers/Present_and_Future_Xprobe2-v1.0.pdf – 1 August 2006.

Beardsley, T. (2003) Snacktime: A Perl Solution for Remote OS Fingerprinting.

http://www.planb-security.net/wp/snacktime.html – 5 September 2006.

Burgess, M. (2006) In: Principles of Network and System Administration, 2nd edition, pages 67, John Wiley & Sons, Ltd, England.

Comer, D.E. (2004) In: Computer Networks and Internets with Internet Applications, 4th edition, Pearson Prentice Hall, NJ, USA.

Contra, A. & Deering, S. (1998) Internet Control Message Protocol

(ICMPv6) for the Internet Protocol Version 6 (IPv6), RFC 2463.

daemon9 (1996) Phrack Magazine, Volume 7, Issue 49, File 06 of 16.

Deering, S. & Hinden, R. (1998) Internet Protocol, Version 6 (IPv6) Specifications, RFC 2460.

Dr. K (2000) In: A Complete H@cker's Handbook, Second Edition, Carlton Books Limited, London.

Fyodor (1998) Remote OS detection via TCP/IP stack Fingerprinting, http://www.insecure.org/nmap/nmap-fingerprinting-article.html – 15 July 2006.

Jones, S. (2003) Port 0 OS Fingerprinting, NetworkPenetration.com. http://networkpenetration.com/port0.html – 23 July 2006

Kent, S. & Atkinson, R. (1998a) IP Authentication Header, RFC 2402.

Kent, S. & Atkinson, R. (1998b) IP Encapsulating Security Payload (ESP), RFC 2406.

Maimon, U. (1996) Phrack Magazine, Volume 7, Issue 49, File 15 of 16.

McClure, S., Scambray, J., Kurtz, G. (2005) In: Hacking Exposed – Network Security Secrets & Solutions, 5th Edition, pages 41 – 76 and back page, McGraw-Hill, California (USA).

Nostromo (2005) Techniques in OS-Fingerprinting. http://nostromo.joeh.org/osf.pdf – 15 August 2006.

Postel, J. (1980) User Datagram Protocol, RFC 768.

Postel, J. (1981a) Transmission Control Protocol, RFC 793.

Postel, J. (1981b) Internet Protocol, RFC 791.

Postel, J. (1981c) Internet Control Message Protocol, RFC 792.

Schneier, B. (2000) Secrets and Lies – Digital Security in a Networked World, Wiley Publishing Inc., Indianapolis, Indiana, USA

Spangler, R. (2003) Analysis of Remote Active Operating System Fingerprinting Tools. http://www.packetwatch.net/documents/papers/osdetection.pdf – 15 August 2006.

Stallings, W. (2004a) Data and Computer Communications, Pearson Prentice Hall, Upper Saddle River, NJ, USA.

Stallings, W. (2004b) In: Computer Networking with Internet Protocols and Technology, Pearson Prentice Hall, Upper Saddle River, NJ, USA.

Peterson, L., Davie, B (2003) Computer Networks – A Systems Approach, Morgan Kaufmann Publishers, San Francisco, CA.

Stopforth, R., Vorster, L., Erwin, D. (2007) Counter-measures for operating system fingerprinting. In: EngineerIT – Electronics, computer, information & communications technology in engineering, page 18. South African Institution for Electrical Engineering. In collaboration with Computer Society of South Africa (CSSA), National Laboratory Association (NLA), IER South Africa Network, IEEE South African Section, CSIR National Metrology Laboratory (NML) and South African Council for Automation and Computation (SACAC).

Wang, K. (2004) Frustrating OS Fingerprinting with Morph.

http://www.synacklabs.net/projects/morph/Wang-Morph-DEFCON12.pdf – 10 August 2006.

Zalewski, M. (2004) Passive OS Fingerprinting Tool version 2.0.4.

http://lcamtuf.coredump.cx/p0f.shtml – 15 August 2006.

Appendices

Appendix A – Protocols

Appendix A.1 Format of the TCP Header

The TCP header contains information about the packet. It has the format shown in Figure A.1.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	28	30	31	32
Source Port																Destination Port															
Sequence Number																															
Acknowledgement Number																															
Data Offset				Reserved						URG	ACK	PSH	RST	SYN	FIN	Window															
Checksum																Urgent Pointer															
Options																								Padding							
Data																															

Figure A.1 – TCP Header

The purpose of each field is as follows:

Source Port (16 bits): This is the source port number.

Destination Port (16 bits): This is the destination port number.

Sequence Number (32 bits): The sequence number will be present in this field, unless a SYN is present. Should a SYN be present, then this field will contain the initial sequence number (ISN). The first data packet that is sent will have this field with a value of ISN+1.

Acknowledgement Number (32 bits): If the ACK control flag is set, this field will contain the sequence number that the sender of the packet is expecting to receive next. This number is always sent once a connection is established.

Data Offset (4 bits): This value indicates how many 32 bits words the header contains. This gives an indication as to where the data begins.

Reserved (6-bits): This field is reserved for future use and should always be zero.

Control Bits (6 bits):

URG:	Urgent Pointer field
ACK:	Acknowledgement field
PSH:	Push Function
RST:	Reset the connection
SYN:	Synchronise sequence numbers
FIN:	No more data from sender

Window (16 bits): This field indicates the number of data octets, starting with the one indicated in the acknowledgement field, that the sender of the packet is willing to receive.

Checksum (16 bits): The checksum is the one's complement of the sum of all 16 bit words in the header and data of the TCP as well as of the 96 bit pseudo header. The pseudo header consists of the source address, the destination address, the protocol and the TCP length as shown in Figure

A.2.

Source Address		
Destination Address		
Zero	Protocol	TCP Length

Figure A.2 – The Pseudo Header

The TCP length is the TCP header length and the data length, but excluding the 12 octets of the pseudo header.

Urgent Pointer (16 bits): This field contains the value of a positive offset from the sequence number of this packet. The urgent field is only taken into account if the urgent control flag is set.

Options (variable length): The options are at the end of the header, and are a multiple of 8 bits. There are two formats that the option field could be:

i) a single octet of option-kind, or,

ii) an octet of option-kind and an octet of option-length and the octets of the actual option-data. Option-length takes into account the entire option field.

Padding (variable length): The padding, which consists of zeroes, is used to ensure that both the TCP header ends and the data begins on a 32-bit boundary.

Appendix A.2 Format of the IPv4 Header

The IPv4 header is shown in Figure A.3.

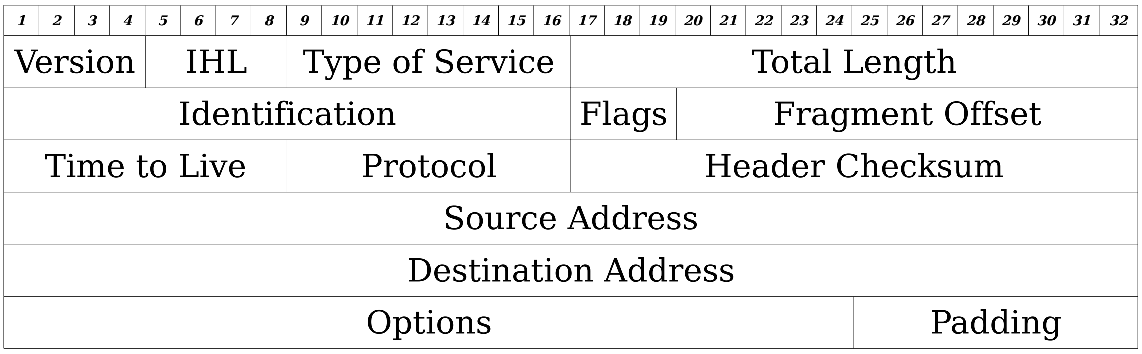


Figure A.3 – Header of IPv4 packet

The purpose of each field is the following:

Version (4 bits): This indicates the version of IP being used, i.e., version 4.

IHL (4 bits): Internet Header Length is the length of the header in words of 32 bits. The minimum value for this field is five.

Type of Service (8 bits): This sets the priority of the packet. The trade off is between low-delay, high-reliability, and high-throughput. These bits are set for the following conditions:

Bits 0 – 2 **:** This is used for the Precedence. The following code implies the corresponding conditions:

111: Network Control

110 :	Internetwork Control
101 :	CRITIC / ECP
100 :	Flash Override
011 :	Flash
010 :	Immediate
001 :	Priority
000 :	Routine

Bit 3	**:** 0 = Normal Delay; 1 = Low Delay
Bit 4	**:** 0 = Normal Throughput; 1 = High Throughput
Bit 5	**:** 0 = Normal Reliability; 1 = High Reliability
Bits 6 – 7	**:** Reserved for future use.

It must be noted that usually two of the three trade offs are set and that the Network control precedence is only used within a network.

Total Length (16 bits): This is the length of the packet (including the header) measured in bytes. The field can have a value of 65 535 bytes, which is impractical to send across a network. It is recommended that before a sender sends a packet that is larger than 576 bytes it must be determined that the receiver will be able to receive that size packet. The value of 576 bytes is chosen as to allow a packet with 512 bytes of data and 64 bytes for the header to be sent.

Identification (16 bits): This field differentiates between the fragmented packets that will be assembled at the receiver.

Flags (3 bits): The bits are the control flags for the following:

Bit 0: reserved and must always be zero.	
Bit 1: (DF)	0 = May Fragment; 1 = Don't Fragment
Bit 2: (MF)	0 = Last Fragment; 1 = More Fragments

Fragment Offset (13 bits): This field indicates where in the packet the fragment belongs. It is measured in units of 8 bytes.

Time to Live (8 bits): The Time to Live field indicates the life span of the packet. Each time the packet goes through a gateway or a router, this field is decremented. The packet is discarded when this field becomes zero. An estimation for this field is one unit for every second that the packet should survive.

Protocol (8 bits): This indicates the protocol used in the data portion of the packet. An example of a protocol value used in this field is determined by differentiating between the TCP and the User Datagram Protocol (UDP).

Header Checksum (16 bits): This is a checksum of the header only. As the fields in the header are changed this field is recomputed at every point on the network where the header is processed. The checksum is calculated the same way that the checksum of the TCP packet is calculated.

Source Address (32 bits): The address of the source of the packet.

Destination Address (32 bits): The address where the packet must be delivered.

Options (variable length): This field has two cases that effects its format. The first is where a single byte of the option-type is used; the second where a byte with the option-type is used as well as a byte that describes the option length (which contains the size of the option-type, option length and the option data) and also a byte for the actual option data.

The option-type byte has three fields. One bit indicates a copied flag, two bits indicate the option class and five bits indicate the option number. If the copied flag bit is set, then this option is copied to all fragments where fragmentation has been implemented.

The option classes for the option type byte are:

0 = control

1 = reserved

2 = debugging and measurement

3 = reserved for future use

The following set of internet options have been defined;

Class	***Number***	***Length***	***Description***
0	0	-	This indicates the end of the Option list. It is only one byte in size. It is used at the end of all the options that could be used and when the end of the options do not coincide with the end of the internet header. It can be copied, introduced or deleted on fragmentation.
0	1	-	This indicates no operation and is also one

Class	***Number***	***Length***	***Description***
			byte in size. It is used between options as a boundary. It may be copied, introduced or deleted on fragmentation.
0	2	11	This indicates that the packets carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with the Department of Defence (DoD) requirements. This field is described later. It must be copied on fragmentation and appears mostly once in a packet.
0	3	var	This indicates Loose Source Routing, which is used to route the packets depending on information supplied by the source. It must be copied on fragmentation, and appears mostly once in a packet.
0	9	var	This indicates strict source routing, which also routes packets depending on information supplied by the source. It must be copied on fragmentation, and appears mostly once in a packet.
0	7	var	This indicates the record route which is used to trace the route a packet takes. It is not copied on fragmentation and goes only in the first fragment. It appears mostly once in a packet.
0	8	4	This is used to carry the stream identification in networks that don't support the stream concept. It must be copied on fragmentation and appears mostly once in a packet.
2	4	var	This indicates the Internet Timestamp. It has a

Class	***Number***	***Length***	***Description***
			maximum length of 40 bits. The value of the timestamp is in milliseconds since midnight UT. If the UT is not known for some reason, then any timestamp can be used, provided that the highest bit is set, to indicate that it is not a standard value. A large enough data space is needed to be able to hold all the timestamps. If the timestamps exceed the available space, then no more are inserted and the overflow counter is incremented. The timestamp is not copied on fragmentation and is only in the first fragment. It appears at most once only in a packet.

The Security field is 16 bits in length, and specifies sixteen levels of security, as described in the following:

0000000000000000-	Unclassified
1111000100110101-	Confidential
0111100010011010-	EFTO
1011110001001101-	MMMM
0101111000100110-	PROG
1010111100010011-	Restricted
1101011110001000-	Secret
0110101111000101-	Top Secret
0011010111100010-	Reserved for future use
1001101011110001-	Reserved for future use
0100110101111000-	Reserved for future use
0010010 10111101-	Reserved for future use
0001001101011110-	Reserved for future use

1000100110101111-	Reserved for future use
1100010011010110-	Reserved for future use
1110001001101011-	Reserved for future use

The Compartments field is also 16 bits in length and has an all zero value if the information that is sent is not compartmented. Other values for this field can be obtained from the Defence Intelligence Agency.

The Handling Restriction field is 16 bits in length of which the values are digraphs (Defence Intelligence Agency Manual (DIAM) 65-19, Standard Security Markings).

The Transmission Control Code (TCC) field is 24 bits in length and provides a way set apart from the traffic and identifies controlled communities of interest among subscribers. These values are trigraphs, and are available from HQ DCA Code 530.

Padding (variable length): Zeroes are padded in the header to make it end on a 32 bit boundary.

Appendix A.3 Internet Control Message Protocol (ICMP)

Most of the information in this appendix has been taken from RFC 792 (Postel 1981c), which deals with the Internet Control Message Protocol.

A.3.1 Time Exceeded Message

This message is sent when the TTL value equals zero or there is an error in the fragmentation process, and the packet has not arrived at it's destination. The format of the Time Exceeded Message ICMP is shown in Figure A.4.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Unused																															
Internet Header + 64 bits of Original Datagram																															

Figure A.4 – Time Exceeded Message ICMP

The fields of the Time Exceeded Message ICMP has the following functions.

Type (8 bits): The value of this field is 11.

Code (8 bits): The code has the following values for the respective message:

0 = Time to live exceeded in transit

1 = Fragment reassembly time exceeded

Code 0 is usually received from a gateway, while code 1 is usually received from a host.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Internet Header + 64 bits of Data Datagram: Should the higher level protocol use a port number, it is then taken as part of the first 64 data bits of the original packet's data.

A.3.2 Parameter Problem Message

The Parameter Problem Message ICMP is sent a datagram is being processed and a problem is detected and the packet is discarded.

The format of the Parameter Problem Message ICMP is shown in Figure A.5.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Pointer								Unused																							
Internet Header + 64 bits of Original Data Datagram																															

Figure A.5 – Parameter Problem Message ICMP

Type (8 bits): The value of this field is 12.

Code (8 bits): The code has the following values for the respective message:

0 = Pointer indicates the error

Code 0 is usually received from either a host or a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Pointer (8 bits): Should the code of the message be equal to 0, then this field points to the byte that caused the error.

Internet Header + 64 bits of Data Datagram: Should the higher level protocol use a port number, it is then taken as part of the first 64 data bits of the original packet's data.

A.3.3 Source Quench Message

A Source Quench Message is usually sent when a gateway's buffer is full and it cannot queue any more packets to be forwarded.

The format of the Source Quench Message ICMP is shown in Figure A.6.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Unused																															
Internet Header + 64 bits of Original Data Datagram																															

Figure A.6 – Source Quench Message ICMP

The fields in the Source Quench Message ICMP has the following functions.

Type (8 bits): The value of this field is 4.

Code (8 bits): The value of this field is 0,which is usually received either from a host or a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Pointer (8 bits): Should the code of the message be equal to 0, then this field points to the byte that caused the error.

Internet Header + 64 bits of Data Datagram: Should the higher level protocol use a port number, it is then taken as part of the first 64 data bits of the original packet's data.

A.3.4 Redirect Message

This message is sent to a host when a gateway finds that there is a shorter path via another gateway that the packets can be sent to it's destination.

The format of the Redirect Message ICMP is shown in Figure A.7.

The fields of the Redirect Message ICMP have the following functions.

Type (8 bits): The value of this field is 5.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Gateway Internet Address																															
Internet Header + 64 bits of Original Data Datagram																															

Figure A.7 – Redirect Message ICMP

Code (8 bits): The code has the following values for the respective message:

0 = Redirect datagrams for the network

1 = Redirect datagrams for the host

2 = Redirect datagrams for the type of service and network

3 = redirect datagrams for the type of service and host

Code 0, 1, 2 and 3 is usually received from a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Gateway Internet Address (32 bits): This field contains the address of the gateway that the original datagram's data should be sent to.

Internet Header + 64 bits of Data Datagram: Should the higher level protocol use a port number, it is then taken as part of the first 64 data bits of the original packet's data.

A.3.5 Timestamp or Timestamp Reply Message

The Timestamp and Timestamp Reply messages can be used to determine the time it takes for the packets to travel in the network.

The format of the Timestamp and Timestamp Reply message ICMPs are shown in Figure A.8.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Identifier																Sequence Number															
Originate Timestamp																															
Receive Timestamp																															
Transmit Timestamp																															

Figure A.8 – Timestamp and Timestamp Reply Message ICMP

The fields in the Timestamp and Timestamp Reply Message ICMP has the following functions.

Type (8 bits): The value of this field is 13 for the timestamp messages and 14 for the timestamp reply messages.

Code (8 bits): The code field has a value of 0 for both type of messages and are usually sent by either a host or a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Identifier (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Identifier field of the Echo Reply message must be the same as that in the Echo Request message field.

Sequence Number (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Sequence Number field of the Echo Reply message must be the same as that in the Echo Request message field.

Originate Timestamp (32 bits): This is the time, in milliseconds (since midnight UT) that the sender sent the Echo message.

Receive Timestamp (32 bits): This is the time, in milliseconds (since midnight UT) the received received the Echo message.

Transmit Timestamp (32 bits): This is the time, in milliseconds (since midnight UT) the receiver of the Echo message sends the Echo Reply message back to the sender of the Echo message.

Should the timestamps not be accurate as the UT is not available, then any time can be used for the timestamp, but the high order bits of the timestamp must be set to indicate that it is a non-standard value.

A.3.6 Information Request and Information Reply Message

The information request messages are sent by a host to find out the number of the network it is physically connected to.

The Information Request and Information Reply Message ICMPs have the format shown in Figure A.8.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Identifier																Sequence Number															

Figure A.8 – Information Request and Information Reply Message

The fields of the Information Request and Information Reply messages have the following functions.

Type (8 bits): The value of this field is 15 for the information request messages and 16 for the information reply messages.

Code (8 bits): The code field has a value of 0 for both type of messages and are usually sent by either a host or a gateway.

Checksum (16 bits): The value of this field is the one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For the calculation of the checksum, the checksum field is taken to be zero.

Identifier (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Identifier field of the Echo Reply message must be the same as that in the Echo Request message field.

Sequence Number (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Sequence Number field of the Echo Reply message must be the same as that in the Echo Request message field.

Appendix A.4 Internet Protocol version 6 (IPv6)

Most of the information in this appendix, has been taken from RFC 2460 (Deering & Hinden 1998), which deals with the Internet protocol version 6.

A.4.1 Format of the IPv6 Header

The IPv6 Header is shown in Figure A.9. The purpose of each field is as the following.

Version (4 bits): This specifies the Internet Protocol version (i.e. 6)

Traffic Class (8 bits): This field is used to identify between different classes and priorities of the packet. If a node supports this field, then it should alter it as needed, otherwise ignore it. If the sender cannot support this field, then it should be set to zero.

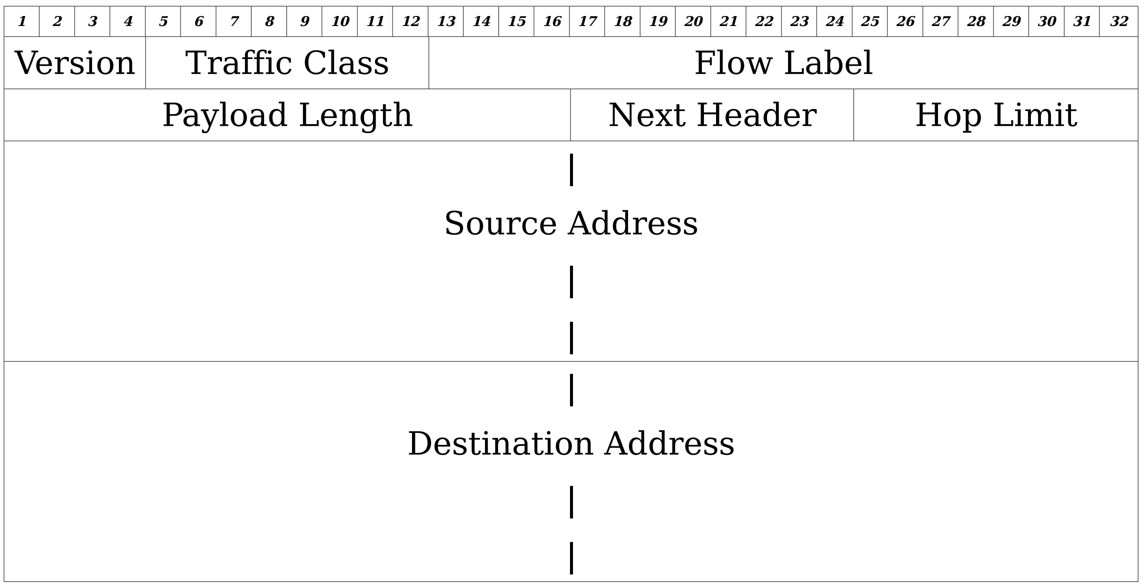


Figure A.9 – IPv6 Header

Flow Label (20 bits): This field is used by the source to label sequences of packets, therefore requesting that special handling is done by the IPv6 routers. Hosts and routers that do not support this field are to be set to zero at the source, or leave unchanged when forwarding the packet and ignore it when receiving the packet.

Payload Length (16 bits): This specifies the length of the payload. This is an indication of the rest of the packet following the IPv6 header in terms of bytes.

Next Header (8 bits): This field identifies the type of header to immediately follow the IPv6 header.

Hop Limit (8 bits): This field's value decrements by one each time the packet is processed by a node. The packet is dropped when a node identifies that the value in this field is equal to zero.

Source Address (128 bits): The address of the original sender of the packet.

Destination Address (128 bits): The address that the packet is to be delivered to.

A.4.1.1 IPv6 Extension Headers

With IPv6 additional headers can be added that are placed between the IPv6 header and that of the header of the layer above it. The number of headers that are used and the type of header used are determined by the Next Header field's value. These extension headers are not processed by any of the nodes that the packet travels along with until the packet has reached the destination address. The destination node then processes the extension headers in the order that they appear.

The only extension header that can be processed by the nodes along which the packet travels is the Hop-by-Hop Option header. This Extension header follows the IPv6 header. The next header field will have a value of zero if this extension header is being used.

Should any node not be able to recognise the Next Header value, the packet is immediately discarded, and an ICMP Parameter Problem message is sent back to the source of the packet with the ICMP Code value of 1. This means that a unrecognised Next Header type has been encountered. The ICMP pointer field will contain the offset of the unrecognised value.

The order of the headers are as follows:

- IPv6 header
- Hop-by-Hop Option header
- Destination Options header, which is processed by the first destination as well as other destinations that are listed in the routing header.
- Routing header
- Fragment header
- Authentication header

- Encapsulating Security Payload header
- Destination Options header, that contains options for only the final destination
- Upper-layer header

Most the headers appear once, except for the destination header which could only appear at most twice. Nodes should be able to process the extension headers in any order if needed, but the Hop-by-Hop header should be first. Even though the nodes are able to process the extension headers in any order, it is recommended that the source puts them in the above order. These headers will be dealt with in further detail.

Before these extension headers are further analysed, it must be noted that the Hop-by-Hop Option header and the Destination Option header both carry different type-length-value (TLV) encoded options. The format is 8 bits for the identifier of the Option Type field, 8 bits for the Option Data Length (which is measured in bytes), and a variable length field for the Option Data. The highest two order bits of the Option Type specifies the action to be taken when an IPv6 node does not recognise the Option Type. These two bits have the following meanings:

00	: skip over this option and continue processing the header.
01	: discard the packet
10	: discard the packet and send an ICMP Parameter Problem with a code 2 message to the packet's source

	address, indicating the unrecognised Option Type.
11	: discard the packet and should the packet's destination address not have been a multicast address, send an ICMP Parameter Problem with a Code 2 message to the packet's source address, indicating the unrecognised Option Type.

The third highest bit of the Option Type indicates if the Option Data of the option can change the en-route of the packet's final destination. Should the Authentication header be present and any option that might change the en-route, then the entire Option Data field must be treated as all zero bytes when verifying the packet's authentication. This third bit has the respective meanings:

0	: Option Data does not change en-route
1	: Option Data may change en-route

A.4.1.1.1 Hop-by-Hop Option Header

This header is used to carry information that must be viewed by every node that has interaction with the packet. It is identified by a Next header value of zero in the IPv6 header. The Hop-by-Hop option header has got the format as shown in Figure A.10.

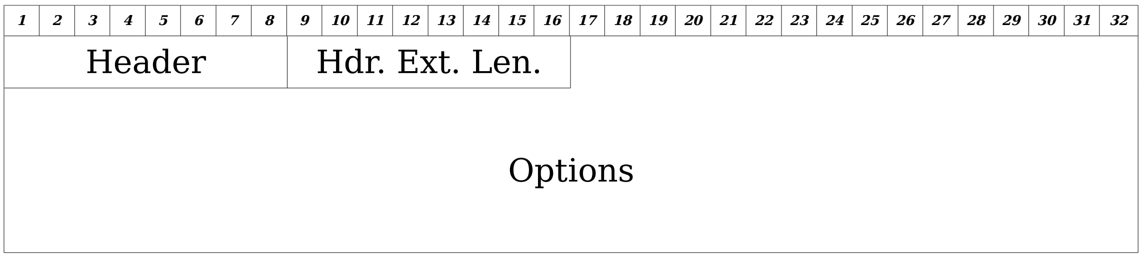


Figure A.10 – Hop-by-Hop Option Header

The fields in the header have the following functions:

Next Header (8 bits): This field identifies the type of header that will follow the Hop-by-Hop option header.

Hdr. Ext. Len. (8 bits): This field indicates the length of the Hop-by-Hop option header, not including the first 8 bytes. The value of this field is in units of eight bytes.

Options (variable length): The length of this field is so that it is a multiple of 8 bytes. It contains at least one TLV-encoded option.

A.4.1.1.2 Routing Header

This header indicates the nodes that the packet must be forwarded by to its destination. This header is identified by having the Next Header field of the previous header as a value of 43. The routing header has the format as shown in Figure A.11.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Next Header								Hdr. Ext. Len.								Routing Type								Segments Left							
type-specific data																															

Figure A.11 – The Routing Header's format

The fields in the Routing Header has the following functions:

Next Header (8 bits): This identifies the the type of header that follows the Routing Header.

Hdr. Ext. Len. (8 bits): This indicates the length of the Routing Header in terms of 8 bytes in length. This excludes the first 8 bytes.

Routing Type (8 bits): This field indicates a particular Routing header variant.

Segment Left (8 bits): This indicates the number of route segments or nodes that are remaining before the packet has reached it's destination.

type-specific data (variable length): The format of this field depends on the Routing Type field. Its length is so that Routing Header is a multiple of 8 bytes.

Certain conditions occur when a received packet is being processed,and it is found that there is an unrecognised Routing Type value. This depends on the Segments Left field. If the Segments Left field is zero, the node must ignore the Routing header and proceed to the next header, which is identified in the Next Header field of the Routing Header. If the Segments Left value is not zero, the node must discard the packet and send an ICMP Parameter Problem, Code 0, message back to the sender.

The Type 0 Routing header has the format shown in Figure A.12.

The fields in the Type 0 Routing header have the following functions:

Next Header (8 bits): This field indicates the type of header that will follow the Type 0 Routing header.

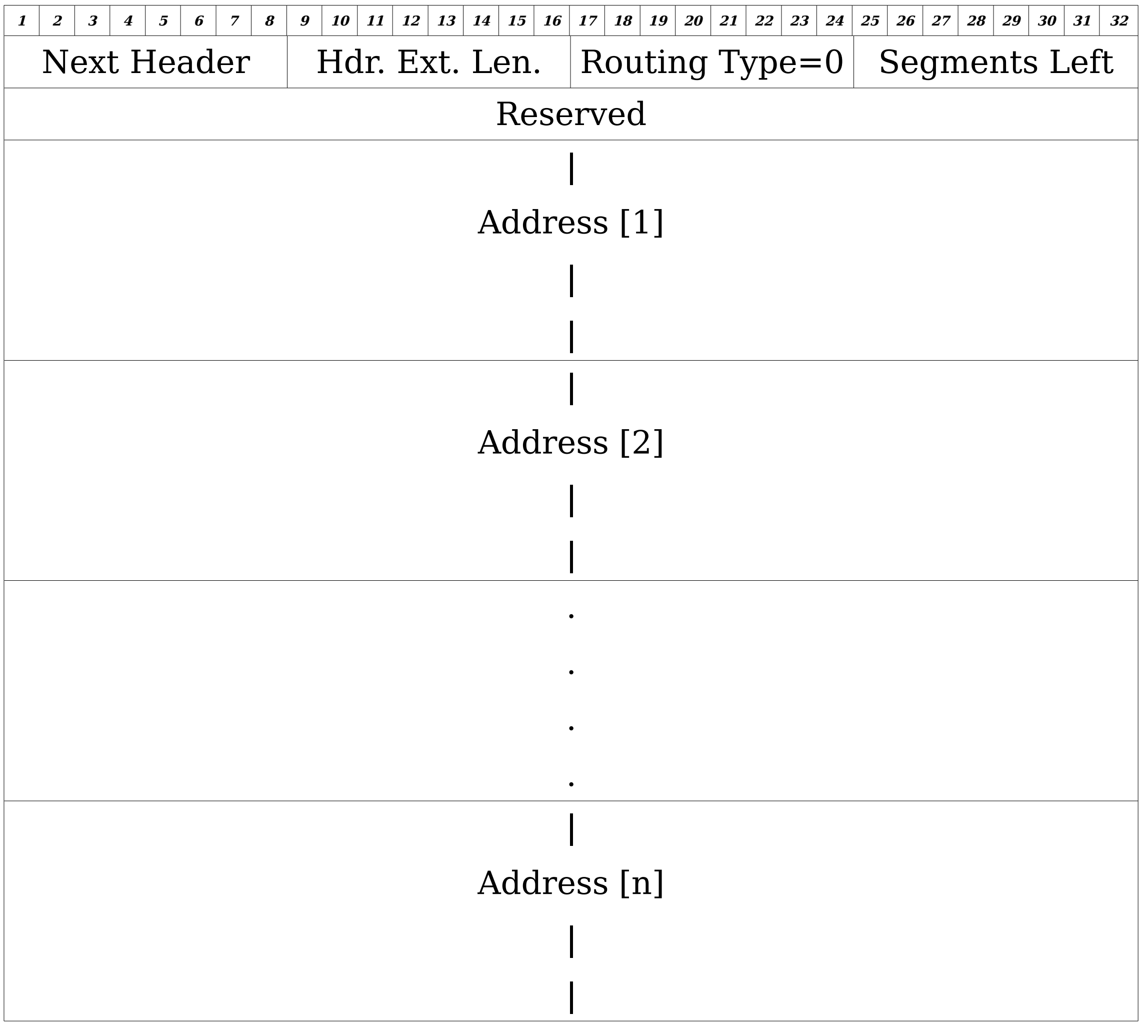


Figure A.12 – Type 0 Routing Header

Hdr. Ext. Len. (8 bits): This is the length of the Routing header in units of 8 bytes. This does not include the first 8 bytes. For the Type 0 Routing header, the Header Extension Length field is equal to twice the number of addresses in the header.

Routing Type (8 bits): In this case this field is equal to zero.

Segment Left (8 bits): This indicates the number of route segments or nodes that remain before the packet has reached it's destination.

Reserved (32 bits): This field is initialised to zero for transmission, but ignored on reception.

Address [1..n] (vectored 128 bits): These addresses are numbered from 1 to n.

Multicasting addresses must not appear in a Type 0 Routing header or in the destination address field of the IPv6 header. The Routing header is not processed by a node unless it has reached the destination address. When it has reached that node, the destination address is swapped with the next address in the Routing header.

A.4.1.1.3 Fragment Header

The fragment header is used when a packet larger than what would fit in the path MTU to the destination, is sent. This is performed only by the source nodes, and not the routers as in IPv4. This header is identified by the Next header field of the previous header with a value of 44. It has the format shown in Figure A.13.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Next Header								Reserved								Fragment Offset													Res		M
Identification																															

Figure A.13 – Fragment Header

The fields in the Fragment header have the following functions:

Next Header (8 bits): This identifies the type of header that follows the Fragment Header.

Reserved (8 bits): This field is reserved. it is initialised to zero on transmission and ignored on reception.

Fragment Offset (13 bits): This field indicates the offset in units of 8 bytes that the data follows after this header, from the start of the Fragmentable Part of the original packet.

Res (2 bits): These two bits are reserved. They are initialised to zero for transmission and ignored on reception.

M flag (1 bit): This is an indication if more fragments are being sent. If it is set to 1, then more fragments are being sent, and if it is set to 0, then this is the last fragment.

Identification (32 bits): This is the identification number that is given to the fragments that come from the same packet that was fragmented. This field is needed so that fragments from other packets are not mixed.

The fragmented packets consist of a header that has the IPv6 header as well as any extended headers that the nodes will process. The fragmented part of the packet consists of the data as well as any headers that are only needed at the destination node.

A.4.1.1.4 Authentication Header

Most of the information in this appendix has been taken from RFC 2402 (Kent & Atkinson 1998a), which deals with the Authentication Header.

The Authentication Header provides data origin authentication for the IPv6 packet. The header is identified by the Next Header field of the previous header that has a value of 51. The Authentication Header has the format shown in Figure A.14.

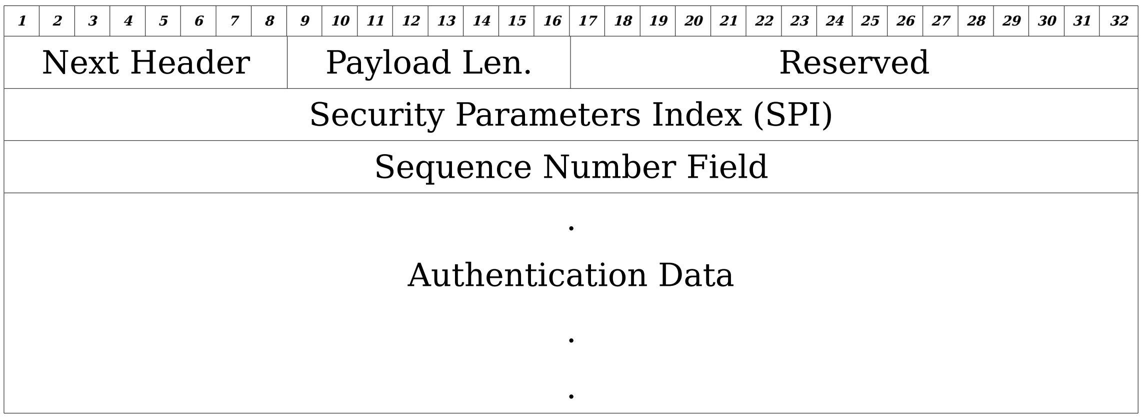


Figure A.14 – Authentication Header

The fields in the Authentication Header have the following functions:

Next Header (8 bits): This identifies the the type of header that follows the Authentication Header.

Payload Len. (8 bits): This field specifies the length of the Authentication

Header in units of 4 bytes, after which 2 of these bytes are subtracted.

Reserved (16 bits): This field is for future use, and should be set to zero.

Security Parameter Index (32 bits): The value of this field is determined by the combination of the destination address as well as the Authentication Header, which identifies the Security Association of the packet. The value from 0 to 255 is reserved by IANA. This field should never be zero.

Sequence Number (32 bits): The value of this field increments for each packet that is sent.

Authentication Data (variable length): This field contains the Integrity Check Value (ICV) of the packet. It must be a multiple of 32 bits in length for IPv4 and 64 bits for IPv6. The ICV is computed from the IP header, the Authentication Header, as well as from the upper layer protocols.

A.4.1.1.5 Encapsulating Security Payload (ESP) Header

Most of the information in this appendix has been taken from RFC 2406 (Kent & Atkinson 1998b), which deals with the ESP Header.

The ESP header is used for data origin authentication, confidentiality and connectionless integrity. This header is identified by the Next Header field of the previous header, with a value of 50. The ESP Header is shown in Figure A.15.

The fields in the ESP packet have the following functions.

Security Parameter Index (32 bits): The value of this field is determined by the combination of the destination address as well as the Authentication Header, which identifies the Security Association of the packet. The value from 0 to 255 are reserved by IANA. This field should never be zero.

Sequence Number (32 bits): The value of this field increments for each packet that is sent.

Payload Data (variable length): This field contains the Initialisation Vector (IV) that is needed when the encryption algorithm needs cryptographic synchronisation data.

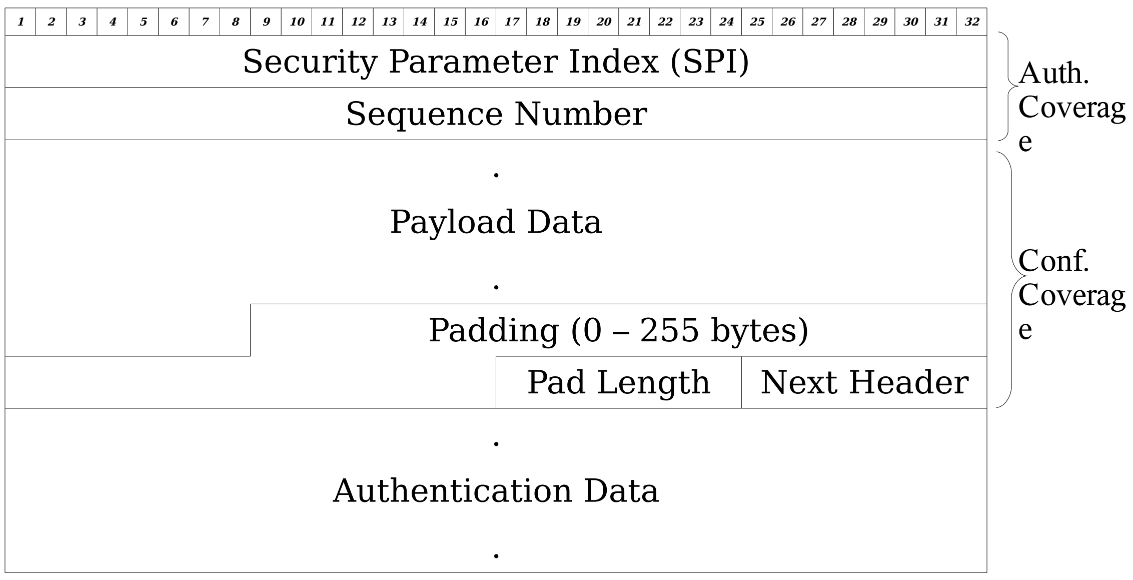


Figure A.15 – ESP Header

Padding (0 – 255 bytes): The padding is needed to fill the plain text so that block sizes of the cipher blocks can be multiples of the block size needed. It is also used so that the cipher text can end on a 4 bytes boundary, as well as to conceal the actual length of the payload. If the encryption algorithm does not specify the padding contents, but padding is needed, then a series of 1 byte sized values, starting from 1, are added, and the following values increment from the previous one. This technique is not only simple, but it also helps against the “cut and paste” attacks as the receiver can check the padding when doing the decryption.

Pad Length (8 bits): This field indicates the padding size of the previous field.

Next Header (8 bits): This identifies the the type of header that follows the ESP Header.

Authentication Data (variable length): This field contains an ICV of the ESP header, excluding the Authentication Data field. Due to the fact that the encryption is not done on the Authentication Data field, the detection of bogus packets is possible before the receiver has performed any decryption of the packet, therefore preventing Denial of Services (DoS) attacks.

A.4.1.1.6 Destination Option Header

The Destination Option header carries optional information that is only processed by the destination node. This header is identified by the Next Header field of the previous header that has a value of 60. The Destination Option header has the format shown in Figure A.16.

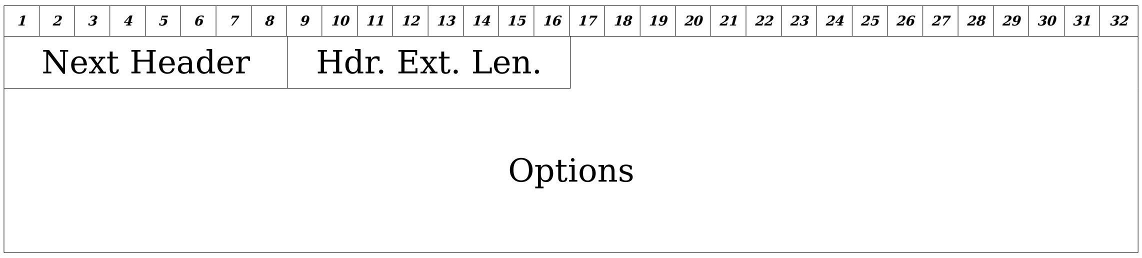


Figure A.16 – Destination Option Header

The fields in the Destination Option header have the following functions:

Next Header (8 bits): This identifies the the type of header that follows the Destination Options Header.

Hdr. Ext. Len. (8 bits): This indicates the length of the Destination Options Header in terms of 8 bytes in length. This excludes the first 8 bytes.

Options (variable length): The length of this field is so that the Destination Options header is a multiple of 8 bytes. It contains at least one TVL-encoded option.

A.4.1.1.7 Upper-Layer Header

These are headers that are used by upper-layer protocols. The pseudo-header shown in Figure A.17 is used for TCP and UDP in IPv6.

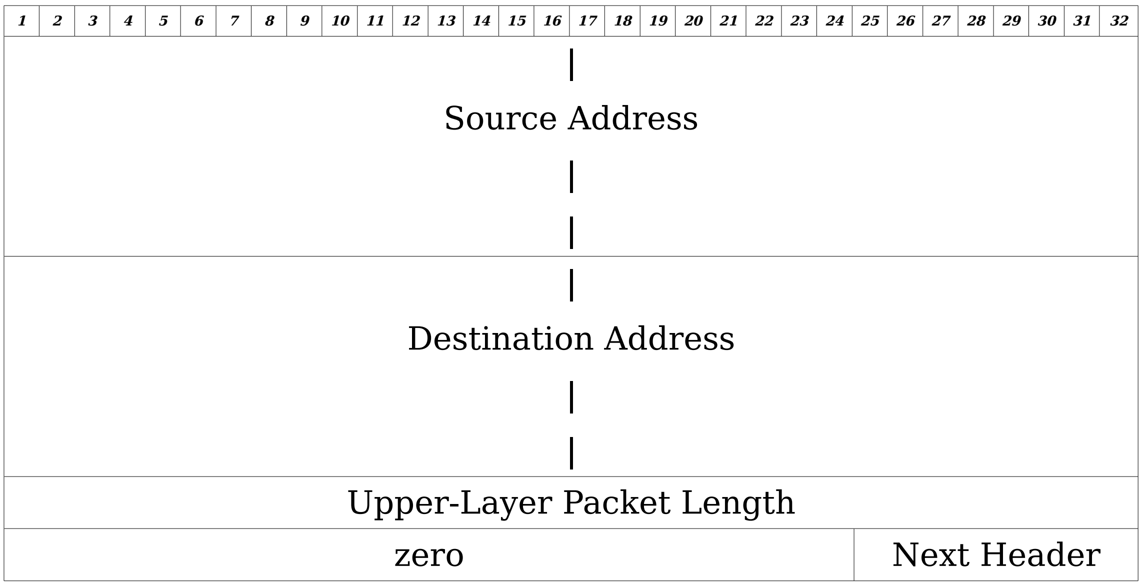


Figure A.17 – Pseudo-header

The fields in the Pseudo-header have the following functions.

Source Address (128 bits): This field has the source address of the TCP packet.

Destination Address (128 bits): This field has the destination address of the TCP packet.

Upper-Layer Packet Length (32 bits): This contains the length of the upper layer packet. Some upper-layer protocols contain the length of the protocol in the packet.

Next Header (8 bits): This identifies the the type of header that follows the pseudo-header.

A.4.1.1.8 No Next Header

This is indicated by the previous header that has the Next Header field with a value of 59.

Appendix A.5 Internet Control Message Protocol for IPv6 (ICMPv6)

Most of the information in this appendix, has been taken from RFC 2463 (Contra & Deering 1998), which deals with the Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6).

As ICMP is taken as part of the IP, it is therefore different for Ipv6. IPv6 nodes used ICMPv6 messages when the packets are processed and errors are encountered. It has a Next Header value of 58.

There are error messages and informational messages. The error messages have message Types from 0 to 127 and the informational messages have message Types from 128 to 255.

ICMPv6 have the following processing rules:

- If the error message type is unknown, it must be passed to the upper layer.

- Should the information message type be unknown, then it is discarded.

- The error message contains as much of the original packet without making the packet exceed the minimum MTU.

- Should the IP need to pass the error message to the upper layers, the protocol type is taken from the original package.

- An ICMP message should not be sent when the following is received:
    - An ICMPv6 message
    - A packet that has an Ipv6 multicast destination address or is sent as a link-layer multicast.
    - A packet that is sent as a link-layer broadcast
    - Should a packet contain a source address that does not identify a single node.
- To preserve bandwidth, the ICMPv6 error messages must be sent at a limited rate. Ways to accomplish this is:
    - To limit the about of error messages in a given time period.
    - To limit the rate that error messages are sent by a fraction of the available bandwidth.

The six different ICMPv6 message formats are discussed below.

A.5.1 Destination Unreachable Error Message

The Destination Unreachable error message is sent when a packet cannot be sent to the destination address. This excludes the reason of congestion.

The format of the Destination Unreachable message is shown in Figure

A.18.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
Unused																															
. As much of the packet that will fit in the ICMP packet, without exceeding the minimum IPv6 MTU. .																															

Figure A.18 – Destination Unreachable Error Message

The fields in the Destination Unreachable message have the following functions:

Type (8 bits): The value of this field is 1.

Code (8 bits): The value of this field has the following meanings:

0 = No route to destination

1 = Communication with Destination administratively prohibited

2 = This value is not yet assigned

3 = Address unreachable

4 = Port unreachable

Checksum (16 bits): This field is the one's complement of the one's complement sum of the entire ICMPv6 message starting with the Type field and ending with the pseudo header. For the calculation of the checksum, the checksum field is taken to be zero.

Unused (32 bits): This field is unused for all code values and the sender should set the value of this field to zero. The receiver should ignore this field.

A.5.2 Packet Too Big Error Message

The Packet Too Big message is sent when the packet is larger than the MTU of a node's outgoing link.

The format of the Packet Too Big message is shown in Figure A.19.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Type								Code								Checksum															
MTU																															
. As much of the packet that will fit in the ICMP packet, without exceeding the minimum IPv6 MTU. .																															

Figure A.19 – Packet Too Big Error Message

The fields in the Packet Too Big message have the following functions:

Type (8 bits): The value of this field is 2.

Code (8 bits): The value of this field is set to zero by the sender. This field is ignored by the receiver.

Checksum (16 bits): This field is the one's complement of the one's complement sum of the entire ICMPv6 message starting with the Type field and ending with the pseudo header. For the calculation of the checksum, the checksum field is taken to be zero.

MTU (32 bits): The Maximum Transmission Unit of the next-hop link.

A.5.3 Time Exceed Error Message

This message is sent when the Hop Limit field has reached zero and the packet is discarded.

The format of the Time Exceed Error Message is shown in Figure A.20.

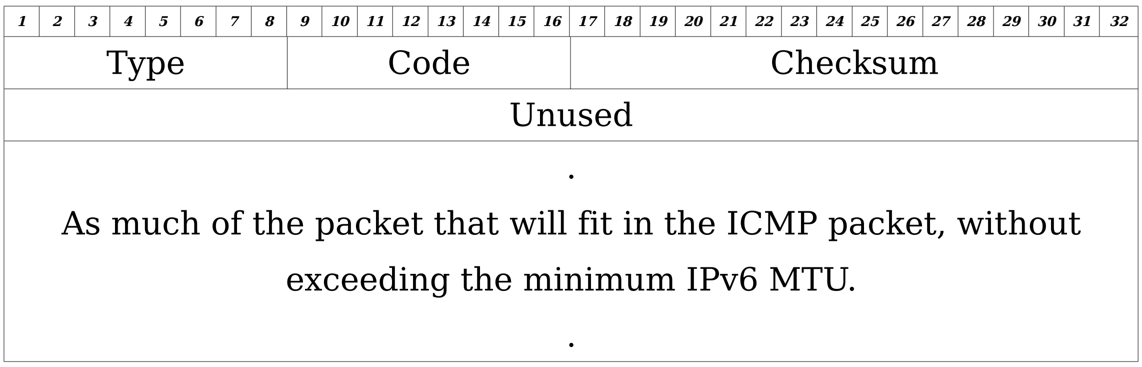


Figure A.20 – Time Exceed Error Message

The functions of the Time Exceed message fields are explained below.

Type (8 bits): The value of this field is 3.

Code (8 bits): The value of this field has the following meanings.

0 = Hop limit exceeded in the transit

1 = Fragment reassembly time exceed

Checksum (16 bits): This field is the one's complement of the one's complement sum of the entire ICMPv6 message starting with the Type field and ending with the pseudo header. For the calculation of the checksum, the checksum field is taken to be zero.

Unused (32 bits): This field is unused for all code values and the sender should set the value of this field to zero. The receiver should ignore this field.

A.5.4 Parameter Problem Error Message

This error is sent when an IPv6 node detects an error in the IPv6 header or extension headers, and the packet is discarded.

The format of the Parameter Problem error message is shown in Figure A.21.

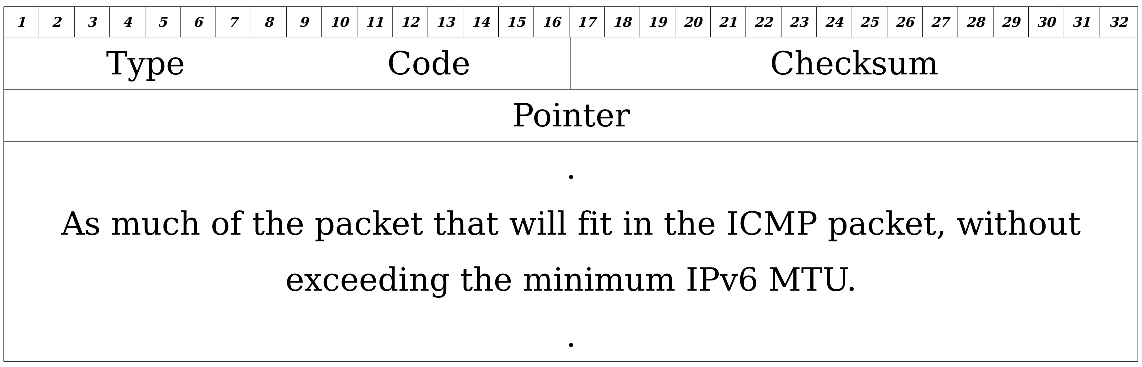


Figure A.21 – Parameter Problem Error Message

The fields of the Parameter Problem Error Message have the following functions:

Type (8 bits): The value of this field is 4.

Code (8 bits): The value of this field has the following meanings:

0 = An error in the header field detected

1 = Unrecognised Next Header type

2 = Unrecognised Ipv6 option

Checksum (16 bits): This field is the one's complement of the one's complement sum of the entire ICMPv6 message starting with the Type field and ending with the pseudo header. For the calculation of the checksum, the checksum field is taken to be zero.

Pointer (32 bits): This field identifies the offset in bytes in the packet in which the error was detected.

A.5.5 Echo Request and Echo Reply Informational Message

The use of this message is for diagnostic purposes. If a node receives an Echo Request ICMPv6, then it should return an Echo Reply ICMPv6.

The format of the Echo Request and Echo Reply informational message is shown in Figure A.22.

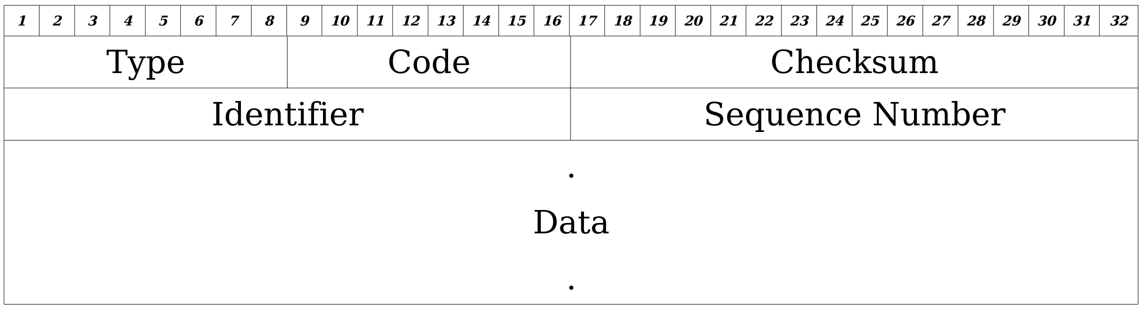


Figure A.22 – Echo Request and Echo Reply Information Message

Type (8 bits): The value of this field is 128 for an Echo Request message, and 129 for Echo Reply message.

Code (8 bits): The value of this field is set to zero by the sender.

Checksum (16 bits): This field is the one's complement of the one's complement sum of the entire ICMPv6 message starting with the Type field and ending with the pseudo header. For the calculation of the checksum, the checksum field is taken to be zero.

Identifier (16 bits): Should the code be equal to zero, then an identifier to aid the matching request and reply messages may be zero. Otherwise, the Identifier field of the Echo Reply message must be the same as that in the Echo Request message field.

Sequence Number (16 bits): Should the code be equal to zero, then an sequence number to aid the matching request and reply messages may be zero. Otherwise the Sequence Number of the Echo Reply message must be the same as that in the Echo Request message field.

Data (variable length): This field contains zero or more bytes of data. The Echo Reply message will have the same data as that in the Echo Request message.

Appendix B – Scripts for firewalls

Appendix B.1 – Settings when firewall is disabled

```
#!/bin/sh


echo "***Initialising No Firewall Rules***";


echo "   - Flush all rules";
/sbin/iptables --flush;
/sbin/iptables --flush INPUT;
/sbin/iptables --flush OUTPUT;
/sbin/iptables --flush FORWARD;
/sbin/iptables --table nat --flush;


echo "   - Delete all chains";
/sbin/iptables --delete-chain;
/sbin/iptables --table nat --delete-chain;


echo "   - Prevent SYN flood attacks";
echo 1 > /proc/sys/net/ipv4/tcp_syncookies;


echo "   - Prevent spoofing source address verification";
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter;


echo "   - Disable response to broadcasts - we don't want to become a smurf
amp";
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts;
echo "   - Enable ICMP redirect acceptance";

echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects;


echo "   - Enable bad error message protection";

echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses;


echo "   - Log spoofed packets, source routed packets, redirect packets";

echo 1 > /proc/sys/net/ipv4/conf/all/log_martians;


echo "   - Enable ICMP echo reply";

echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_all;


echo "   - Modify TOS value of all outgoing packets to 0xc0";

/sbin/iptables -t mangle -I OUTPUT -j TOS --set-tos 0xc0;


echo "***Done***";
```

Appendix B.2 – Settings when firewall is enabled

```
#!/bin/sh


echo "***Initialising Suse 10.1 Firewall Rules***";


echo "   - Flush all rules";
/sbin/iptables --flush;
/sbin/iptables --flush INPUT;
/sbin/iptables --flush OUTPUT;
/sbin/iptables --flush FORWARD;
/sbin/iptables --table nat --flush;


echo "   - Delete all chains";
/sbin/iptables --delete-chain;
/sbin/iptables --table nat --delete-chain;


echo "   - Set default chain policies";
/sbin/iptables --policy INPUT DROP;
/sbin/iptables --policy OUTPUT DROP;
/sbin/iptables --policy FORWARD DROP;
/sbin/iptables --new forward_ext;
/sbin/iptables --new input_ext;
/sbin/iptables --new reject_func;


echo "   - Prevent SYN flood attacks";
echo 1 > /proc/sys/net/ipv4/tcp_syncookies;
echo "   - Prevent spoofing source address verification";
echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter;


echo "   - Disable response to broadcasts - we don't want to become a smurf
amp";
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts;


echo "   - Enable ICMP redirect acceptance";
echo 1 > /proc/sys/net/ipv4/conf/all/accept_redirects;


echo "   - Enable bad error message protection";
echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses;


echo "   - Log spoofed packets, source routed packets, redirect packets";
echo 1 > /proc/sys/net/ipv4/conf/all/log_martians;


echo "   - Enable ICMP echo reply";
echo 0 > /proc/sys/net/ipv4/icmp_echo_ignore_all;


echo "   - Modify TOS value of all outgoing packets to 0xc0";
/sbin/iptables -t mangle -I OUTPUT -j TOS --set-tos 0xc0;


echo "   - Adding Pre-defined Firewall Rules from Suse 10.1";
/sbin/iptables -A INPUT -i lo -j ACCEPT;
/sbin/iptables -A INPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT;
/sbin/iptables -A INPUT -i eth0 -j input_ext;
```

```
/sbin/iptables -A INPUT -j input_ext;

/sbin/iptables -A INPUT -m limit --limit 3/min -j LOG --log-prefix "SFW2-IN-
ILL-TARGET " --log-tcp-options --log-ip-options;

/sbin/iptables -A INPUT -j DROP;

/sbin/iptables -A FORWARD -m limit --limit 3/min -j LOG --log-prefix "SFW2-
FWD-ILL-ROUTING " --log-tcp-options --log-ip-options;

/sbin/iptables -A OUTPUT -o lo -j ACCEPT;

/sbin/iptables -A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j
ACCEPT;

/sbin/iptables -A OUTPUT -m limit --limit 3/min -j LOG --log-prefix "SFW2-
OUT-ERROR " --log-tcp-options --log-ip-options;

/sbin/iptables -A input_ext -m pkttype --pkt-type broadcast -j DROP;

/sbin/iptables -A input_ext -p icmp -m icmp --icmp-type 4 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m icmp --icmp-type 8 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 0 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 3 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 11 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 12 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 14 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 18 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 3/2 -j ACCEPT;

/sbin/iptables -A input_ext -p icmp -m state --state RELATED,ESTABLISHED
-m icmp --icmp-type 5 -j ACCEPT;
```

```
/sbin/iptables -A input_ext -p tcp -m limit --limit 3/min -m tcp --dport 22 --
tcp-flags FIN,SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-ACC-TCP "
--log-tcp-options --log-ip-options;

/sbin/iptables -A input_ext -p tcp -m tcp --dport 22 -j ACCEPT;

/sbin/iptables -A input_ext -p tcp -m tcp --dport 113 -m state --state NEW -j
reject_func;

/sbin/iptables -A input_ext -m limit --limit 3/min -m pkttype --pkt-type
multicast -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --log-tcp-options --
log-ip-options;

/sbin/iptables -A input_ext -m pkttype --pkt-type multicast -j DROP;

/sbin/iptables -A input_ext -p tcp -m limit --limit 3/min -m tcp --tcp-flags
FIN,SYN,RST,ACK SYN -j LOG --log-prefix "SFW2-INext-DROP-DEFLT " --
log-tcp-options --log-ip-options;

/sbin/iptables -A input_ext -p icmp -m limit --limit 3/min -j LOG --log-prefix
"SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options;

/sbin/iptables -A input_ext -p udp -m limit --limit 3/min -j LOG --log-prefix
"SFW2-INext-DROP-DEFLT " --log-tcp-options --log-ip-options;

/sbin/iptables -A input_ext -m limit --limit 3/min -m state --state INVALID -j
LOG --log-prefix "SFW2-INext-DROP-DEFLT-INV " --log-tcp-options --log-ip-
options;

/sbin/iptables -A input_ext -j DROP;

/sbin/iptables -A reject_func -p tcp -j REJECT --reject-with tcp-reset;

/sbin/iptables -A reject_func -p udp -j REJECT --reject-with icmp-port-
unreachable;

/sbin/iptables -A reject_func -j REJECT --reject-with icmp-proto-
unreachable;


echo "***Done***";
```

Appendix B.3 – Suggested Additions to firewalls

```
#!/bin/sh

echo "***Initialising Suggested Firewall Rules***";


echo "   - Prevent SYN flood attacks";

echo 1 > /proc/sys/net/ipv4/tcp_syncookies;


echo "   - Prevent spoofing source address verification";

echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter;


echo "   - Disable response to broadcasts - we don't want to become a smurf
amp";

echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts;


echo "   - Disable ICMP redirect acceptance";

echo 0 > /proc/sys/net/ipv4/conf/all/accept_redirects;


echo "   - Enable bad error message protection";

echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses;


echo "   - Log spoofed packets, source routed packets, redirect packets";

echo 1 > /proc/sys/net/ipv4/conf/all/log_martians;


echo "   - Disable ICMP echo reply";

echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all;
echo "   - Drop outgoing ICMP";
/sbin/iptables -p icmp --icmp-type any -I OUTPUT -j DROP;

echo "   - Modify TOS value of all outgoing packets to 0x00";
/sbin/iptables -t mangle -I OUTPUT -j TOS --set-tos 0x00;

echo "   - Drop all packets with port 0";
/sbin/iptables -I INPUT -p tcp --dport 0 -j DROP;
/sbin/iptables -I INPUT -p udp --dport 0 -j DROP;
/sbin/iptables -I INPUT -p tcp --sport 0 -j DROP;
/sbin/iptables -I INPUT -p udp --sport 0 -j DROP;

echo "   - Initiating OSF module";
/sbin/insmod /home/user/osf/ipt_osf.ko;
/home/user/osf/load /home/user/osf/pf.os /proc/sys/net/ipv4/osf;
/sbin/iptables -I INPUT -p tcp -m osf --genre Nmap --log 2  --ttl 2 -j DROP;

echo "***Done***";
```

Appendix C – Results of Tests and Experiments with IPv4

Appendix C.1 – Output of the Port Scanning techniques

SYN Scan

The firewall was disabled and the following output was received from Nmap:

```
linux-:/# nmap -sS -vv  10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:32 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:32
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0, TR: 3, CN: 0]
Initiating SYN Stealth Scan against 10.0.0.6 [1672 ports] at 11:32
Discovered open port 22/tcp on 10.0.0.6
Discovered open port 111/tcp on 10.0.0.6
Discovered open port 631/tcp on 10.0.0.6
The SYN Stealth Scan took 1.81s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
```

```
22/tcp  open  ssh
111/tcp open  rpcbind
631/tcp open  ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 16.560 seconds
               Raw packets sent: 1680 (67.2KB) | Rcvd: 1673 (77KB)
```

The firewall was then enabled on system B and the following output was observed.

```
linux:/# nmap -sS -vv  10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:13 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:13
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating SYN Stealth Scan against 10.0.0.6 [1672 ports] at 11:13
Discovered open port 22/tcp on 10.0.0.6
The SYN Stealth Scan took 21.39s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)
```

```
Nmap finished: 1 IP address (1 host up) scanned in 35.297 seconds
               Raw packets sent: 3347 (134KB) | Rcvd: 7 (318B)
```

UDP Scan

The following output was given by Nmap when the firewall was disabled:

```
linux:/# nmap -sU -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:33 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:33
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating UDP Scan against 10.0.0.6 [1482 ports] at 11:33
Increasing send delay for 10.0.0.6 from 0 to 50 due to max_successful_tryno
increase to 4
Increasing send delay for 10.0.0.6 from 50 to 100 due to max_successful_tryno
increase to 5
Increasing send delay for 10.0.0.6 from 100 to 200 due to max_successful_tryno
increase to 6
Increasing send delay for 10.0.0.6 from 200 to 400 due to 11 out of 11 dropped
probes since last increase.
Increasing send delay for 10.0.0.6 from 400 to 800 due to 11 out of 11 dropped
probes since last increase.
UDP Scan Timing: About 3.21% done; ETC: 11:49 (0:15:25 remaining)
UDP Scan Timing: About 67.46% done; ETC: 11:57 (0:07:42 remaining)
The UDP Scan took 1476.42s to scan 1482 total ports.
Host 10.0.0.6 appears to be up ... good.
```

```
Interesting ports on 10.0.0.6:
(The 1478 ports scanned but not shown below are in state: closed)
PORT     STATE         SERVICE
68/udp   open|filtered dhcpc
111/udp  open|filtered rpcbind
631/udp  open|filtered unknown
1024/udp open|filtered unknown
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 1490.257 seconds
               Raw packets sent: 1934 (54.2KB) | Rcvd: 1483 (83KB)
```

The following output was observed when the firewall was enabled:

```
linux:/# nmap -sU -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:14 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:14
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating UDP Scan against 10.0.0.6 [1482 ports] at 11:15
The UDP Scan took 31.91s to scan 1482 total ports.
Host 10.0.0.6 appears to be up ... good.
All 1482 scanned ports on 10.0.0.6 are: open|filtered
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 45.705 seconds
               Raw packets sent: 2965 (83KB) | Rcvd: 1 (42B)
```

TCP Scan

With the firewall disabled, the following output was given by Nmap:

```
linux:/# nmap -sT -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:59 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:59
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating Connect() Scan against 10.0.0.6 [1672 ports] at 11:59
Discovered open port 631/tcp on 10.0.0.6
Discovered open port 22/tcp on 10.0.0.6
Discovered open port 111/tcp on 10.0.0.6
The Connect() Scan took 2.55s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
631/tcp open  ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 16.882 seconds
               Raw packets sent: 1 (42B) | Rcvd: 1 (42B)
```

When the firewall was disabled, the following output was observed.

```
linux:/# nmap -sT -vv --dns_servers 10.0.0.0 10.0.0.6


Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:16 SAST

Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:16

The ARP Ping Scan took 0.01s to scan 1 total hosts.

DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating Connect() Scan against 10.0.0.6 [1672 ports] at 11:16

Discovered open port 22/tcp on 10.0.0.6

The Connect() Scan took 30.89s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

Interesting ports on 10.0.0.6:

(The 1670 ports scanned but not shown below are in state: filtered)

PORT    STATE  SERVICE

22/tcp  open   ssh

113/tcp closed auth

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 44.588 seconds

              Raw packets sent: 1 (42B) | Rcvd: 1 (42B)
```

Null Scan

The following output was observed when the firewall was disabled:

```
linux:/# nmap -sN -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:00 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:00
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating NULL Scan against 10.0.0.6 [1672 ports] at 12:00
The NULL Scan took 1.54s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE         SERVICE
22/tcp  open|filtered ssh
111/tcp open|filtered rpcbind
631/tcp open|filtered ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 15.802 seconds
               Raw packets sent: 1676 (67KB) | Rcvd: 1670 (76.8KB)
```

The following was observed when the firewall was enabled:

```
linux:/# nmap -sN -vv --dns_servers 10.0.0.0 10.0.0.6
```

```
Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:17 SAST

Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:17

The ARP Ping Scan took 0.01s to scan 1 total hosts.

DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating NULL Scan against 10.0.0.6 [1672 ports] at 11:18

The NULL Scan took 35.86s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

All 1672 scanned ports on 10.0.0.6 are: open|filtered

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 49.631 seconds

                Raw packets sent: 3345 (134KB) | Rcvd: 1 (42B)
```

ACK Scan

With the firewall disabled the following output was given by Nmap:

```
linux:/# nmap -sA -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:01 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:01
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating ACK Scan against 10.0.0.6 [1672 ports] at 12:01
The ACK Scan took 1.06s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
All 1672 scanned ports on 10.0.0.6 are: UNfiltered
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 15.474 seconds
               Raw packets sent: 1673 (66.9KB) | Rcvd: 1673 (77KB)
```

On the contrary, when the firewall was enabled, the following output was observed:

```
linux:/# nmap -sA -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:19 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:19
The ARP Ping Scan took 0.01s to scan 1 total hosts.
```

```
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating ACK Scan against 10.0.0.6 [1672 ports] at 11:19

The ACK Scan took 22.04s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

Interesting ports on 10.0.0.6:

(The 1670 ports scanned but not shown below are in state: filtered)

PORT    STATE      SERVICE

22/tcp  UNfiltered ssh

113/tcp UNfiltered auth

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 35.982 seconds

             Raw packets sent: 3347 (134KB) | Rcvd: 7 (318B)
```

FIN Scan

With the firewall disabled the following output was given by Nmap:

```
linux:/# nmap -sF -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:02 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:02
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating FIN Scan against 10.0.0.6 [1672 ports] at 12:03
The FIN Scan took 2.70s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE         SERVICE
22/tcp  open|filtered ssh
111/tcp open|filtered rpcbind
631/tcp open|filtered ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 17.011 seconds
             Raw packets sent: 1685 (67.4KB) | Rcvd: 1670 (76.8KB)
```

With the firewall enabled the following output was observed:

```
linux:/# nmap -sF -vv --dns_servers 10.0.0.0 10.0.0.6
```

```
Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:20 SAST

Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:20

The ARP Ping Scan took 0.02s to scan 1 total hosts.

DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating FIN Scan against 10.0.0.6 [1672 ports] at 11:20

The FIN Scan took 35.89s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

All 1672 scanned ports on 10.0.0.6 are: open|filtered

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 49.662 seconds

              Raw packets sent: 3345 (134KB) | Rcvd: 1 (42B)
```

Window Scan

The following output was seen when the firewall was disabled:

```
linux:/# nmap -sW -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:03 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:03
The ARP Ping Scan took 0.04s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating Window Scan against 10.0.0.6 [1672 ports] at 12:04
The Window Scan took 0.85s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
All 1672 scanned ports on 10.0.0.6 are: closed
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 15.698 seconds
           Raw packets sent: 1673 (66.9KB) | Rcvd: 1673 (77KB)
```

When the firewall was enabled the following output was observed:

```
linux:/# nmap -sW -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:22 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:22
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating Window Scan against 10.0.0.6 [1672 ports] at 11:22

The Window Scan took 22.01s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

Interesting ports on 10.0.0.6:

(The 1670 ports scanned but not shown below are in state: filtered)

PORT    STATE  SERVICE

22/tcp  closed ssh

113/tcp closed auth

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 35.861 seconds

              Raw packets sent: 3347 (134KB) | Rcvd: 7 (318B)
```

Xmas Scan

The following output was received from Nmap when the firewall was disabled:

```
linux:/# nmap -sX -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:04 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:04
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating XMAS Scan against 10.0.0.6 [1672 ports] at 12:05
The XMAS Scan took 1.95s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE         SERVICE
22/tcp  open|filtered ssh
111/tcp open|filtered rpcbind
631/tcp open|filtered ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 16.268 seconds
               Raw packets sent: 1676 (67KB) | Rcvd: 1670 (76.8KB)
```

The following output was observed when the firewall was enabled:

```
linux:/# nmap -sX -vv --dns_servers 10.0.0.0 10.0.0.6


Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:23 SAST

Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:23

The ARP Ping Scan took 0.02s to scan 1 total hosts.

DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating XMAS Scan against 10.0.0.6 [1672 ports] at 11:23

The XMAS Scan took 35.83s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

All 1672 scanned ports on 10.0.0.6 are: open|filtered

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 49.682 seconds

               Raw packets sent: 3345 (134KB) | Rcvd: 1 (42B)
```

TCP Maimon Scan

With the firewall disabled, the following output was received from Nmap:

```
linux:/# nmap -sM -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:05 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:05
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating Maimon Scan against 10.0.0.6 [1672 ports] at 12:06
The Maimon Scan took 0.86s to scan 1672 total ports.
Host 10.0.0.6 appears to be up ... good.
All 1672 scanned ports on 10.0.0.6 are: closed
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 15.159 seconds
          Raw packets sent: 1673 (66.9KB) | Rcvd: 1673 (77KB)
```

With the firewall disabled the following output was observed:

```
linux:/# nmap -sM -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:25 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:25
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]

Initiating Maimon Scan against 10.0.0.6 [1672 ports] at 11:25

The Maimon Scan took 22.24s to scan 1672 total ports.

Host 10.0.0.6 appears to be up ... good.

Interesting ports on 10.0.0.6:

(The 1671 ports scanned but not shown below are in state: open|filtered)

PORT   STATE  SERVICE

22/tcp closed ssh

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 36.180 seconds

             Raw packets sent: 3348 (134KB) | Rcvd: 6 (272B)
```

Protocol Scan

The following output was received when the firewall was disabled:

```
linux:/# nmap -sO -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 12:06 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 12:06
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.01s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating IPProto Scan against 10.0.0.6 [256 ports] at 12:06
Increasing send delay for 10.0.0.6 from 0 to 5 due to max_successful_tryno
increase to 4
Increasing send delay for 10.0.0.6 from 5 to 10 due to max_successful_tryno
increase to 5
Increasing send delay for 10.0.0.6 from 10 to 20 due to max_successful_tryno
increase to 6
Increasing send delay for 10.0.0.6 from 20 to 40 due to max_successful_tryno
increase to 7
Increasing send delay for 10.0.0.6 from 40 to 80 due to max_successful_tryno
increase to 8
Increasing send delay for 10.0.0.6 from 80 to 160 due to max_successful_tryno
increase to 9
Increasing send delay for 10.0.0.6 from 160 to 320 due to 11 out of 12 dropped
probes since last increase.
IPProto Scan Timing: About 18.43% done; ETC: 12:09 (0:02:14 remaining)
Increasing send delay for 10.0.0.6 from 320 to 640 due to 11 out of 11 dropped
probes since last increase.
Increasing send delay for 10.0.0.6 from 640 to 1000 due to 11 out of 19 dropped
```

```
probes since last increase.
Discovered open port 6/ip on 10.0.0.6
Discovered open port 1/ip on 10.0.0.6
IPProto Scan Timing: About 67.22% done; ETC: 12:11 (0:01:20 remaining)
The IPProto Scan took 274.10s to scan 256 total ports.
Host 10.0.0.6 appears to be up ... good.
Interesting protocols on 10.0.0.6:
(The 251 protocols scanned but not shown below are in state: closed)
PROTOCOL STATE         SERVICE
1        open          icmp
2        open|filtered igmp
6        open          tcp
17       filtered      udp
41       open|filtered ipv6
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)

Nmap finished: 1 IP address (1 host up) scanned in 287.904 seconds
               Raw packets sent: 433 (8718B) | Rcvd: 257 (12.3KB)
```

With the firewall enabled the following output was observed:

```
linux:/# nmap -sO -vv --dns_servers 10.0.0.0 10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:26 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:26
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating IPProto Scan against 10.0.0.6 [256 ports] at 11:26
```

```
Discovered open port 1/ip on 10.0.0.6

The IPProto Scan took 6.45s to scan 256 total ports.

Host 10.0.0.6 appears to be up ... good.

Interesting protocols on 10.0.0.6:

(The 255 protocols scanned but not shown below are in state: open|filtered)

PROTOCOL STATE SERVICE

1        open  icmp

MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)


Nmap finished: 1 IP address (1 host up) scanned in 20.264 seconds

              Raw packets sent: 512 (10.3KB) | Rcvd: 2 (88B)
```

Appendix C.2 – OS Detection

Nmap's Results

Two computer systems were set up to demonstrate the scans. The computers used had the following configurations:

Computer A:

IP Address: 10.0.0.5

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Computer B:

IP Address: 10.0.0.6

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Opened ports/services: port 22 / SSH

System A was used to do the scans on system B, with the scenario that system B has the firewall disabled and enabled respectively. The following results were observed in the scans.

Firstly, the firewall is disabled. Nmap's output was the following:

```
linux:/# nmap -O -vv  10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:07 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:07
The ARP Ping Scan took 0.02s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0,
TR: 3, CN: 0]
Initiating SYN Stealth Scan against 10.0.0.6 [1672 ports] at 11:07
Discovered open port 22/tcp on 10.0.0.6
Discovered open port 631/tcp on 10.0.0.6
Discovered open port 111/tcp on 10.0.0.6
The SYN Stealth Scan took 0.48s to scan 1672 total ports.
For OSScan assuming port 22 is open, 1 is closed, and neither are firewalled
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1669 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
631/tcp open  ipp
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.7 - 2.6.11
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=38C174%IPID=Z)
T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UC
K=E%ULEN=134%DAT=E)
```

```
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=3719540 (Good luck!)
IPID Sequence Generation: All zeros

Nmap finished: 1 IP address (1 host up) scanned in 16.292 seconds
               Raw packets sent: 1688 (68.1KB) | Rcvd: 1686 (78KB)
```

The firewall on system B then was enabled and the following output was observed:

```
linux:/# nmap -O -vv  10.0.0.6

Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-07-17 11:05 SAST
Initiating ARP Ping Scan against 10.0.0.6 [1 port] at 11:05
The ARP Ping Scan took 0.01s to scan 1 total hosts.
DNS resolution of 1 IPs took 13.00s. Mode: Async [#: 1, OK: 0, NX: 0, DR: 1, SF: 0, TR: 3, CN: 0]
Initiating SYN Stealth Scan against 10.0.0.6 [1672 ports] at 11:05
Discovered open port 22/tcp on 10.0.0.6
The SYN Stealth Scan took 21.38s to scan 1672 total ports.
For OSScan assuming port 22 is open, 113 is closed, and neither are firewalled
Host 10.0.0.6 appears to be up ... good.
Interesting ports on 10.0.0.6:
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE  SERVICE
22/tcp  open   ssh
113/tcp closed auth
MAC Address: 00:0E:A6:73:E7:25 (Asustek Computer)
Device type: general purpose|broadband router
```

```
Running: Linux 2.4.X|2.5.X|2.6.X, D-Link embedded

OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.18 - 2.4.20, Linux 2.4.26, Linux 2.4.27 or
D-Link DSL-500T (running linux 2.4), Linux 2.4.7 - 2.6.11, Linux 2.6.0 - 2.6.11

OS Fingerprint:

TSeq(Class=RI%gcd=1%SI=39A613%IPID=Z)

T1(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)

T2(Resp=N)

T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)

T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)

T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)

T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)

T7(Resp=N)

PU(Resp=N)


TCP Sequence Prediction: Class=random positive increments

                         Difficulty=3778067 (Good luck!)

IPID Sequence Generation: All zeros


Nmap finished: 1 IP address (1 host up) scanned in 37.469 seconds

               Raw packets sent: 3364 (135KB) | Rcvd: 18 (936B)
```

Xprobe2's Results

Two computer systems were set up to demonstrate the scans. The computers used had the following configurations:

Computer A:

IP Address: 10.0.0.5

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Computer B:

IP Address: 10.0.0.6

Operating System: SUSE 10.1 running Linux kernel version 2.6.16.13-4

Opened ports/services: port 22 / SSH

System A was used to do the scans on system B, with the scenario that system B has the firewall disabled and enabled respectively. The following results were observed in the scans.

Firstly, the firewall is disabled. Xprobe2's output was the following:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com,
meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
```

```
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting
module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting
module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test
failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module
test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00032 sec
[+] Selected safe Round-Trip Time value is: 0.00063 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip
for www.securityfocus.com in UDP probe
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.22" (Guess probability: 100%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.23" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.21" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.20" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.19" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.24" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.25" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.26" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.27" (Guess probability: 100%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.28" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

The firewall on system B then was enabled and the following output was observed:

```
linux:/# xprobe2 10.0.0.6

Xprobe2 v.0.3 Copyright (c) 2002-2005 fyodor@o0o.nu, ofir@sys-security.com, meder@o0o.nu

[+] Target is 10.0.0.6
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.0.0.6. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.0.0.6. Module test failed
[-] No distance calculation. 10.0.0.6 appears to be dead or no ports known
[+] Host: 10.0.0.6 is up (Guess probability: 50%)
[+] Target: 10.0.0.6 is alive. Round-Trip Time: 0.00202 sec
[+] Selected safe Round-Trip Time value is: 0.00405 sec
[-] icmp_port_unreach::build_DNS_reply(): gethostbyname() failed! Using static ip for www.securityfocus.com in UDP probe
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 03.0.01eTc1" (Guess probability: 100%)
[+] Other guesses:
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.21" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.22" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.04T53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare Version 07.5.05KT53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.01BT51" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.6.04aT51" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Foundry Networks IronWare 07.7.01eT53" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.23" (Guess probability: 91%)
[+] Host 10.0.0.6 Running OS: "Linux Kernel 2.4.24" (Guess probability: 91%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

Appendix D – Installation of the OSF Module

The procedure of installing the OSF module is with the following steps:

1. Make sure that a C compiler is installed on the system. The one that is configured by the OSF module is GCC (GNU C Compiler).

2. Make sure that the Linux source files are on the system. These files are found in the /usr/src/linux/ directory, which needs to be available.

3. The IPtables source files are also needed, especially the iptables.h file and the libiptc/ directory. If these are not available on the system, they should be downloaded and saved in a directory on it. For description purposes it is going to be assumed that a copy of the source files are in the /home/user/iptables directory.

4. The OSF source files must also be available on the system. For description purposes, it is going to be assumed that it is saved in the /home/user/osf direcory.

5. While in the /home/user/osf directory, edit the Makefile file with an editor. The following lines are altered:

   - IPTABLES:= path_to_iptables_sources_or_header_files

     Becomes

     IPTABLES:= /home/user/iptables

- iptables_version=$(shell(/sbin/iptables -V | awk {'print $$2'} | cut -c 2-))

  Becomes

  iptables_version=$(shell(/usr/sbin/iptables -V | awk {'print $$2'} | cut -c 2-))

  This is not a standard modification on all distributions. A search should be performed for the directory that IPtables executable file is stored in. This can be done with the command:

  which iptables

- Errors can occur when trying to compile the OSF module. This is due to the fact that OSF compilation uses different information for version fields than that of IPtables. It has been found that errors occurred in the compilation are solved with the following alterations.

  - KDIR:= /lib/modules/$(shell uname -r)/build

    Becomes

    KDIR:= /lib/modules/<kernel version>/build

    Where <kernel version> is determined by the statement:

    uname -r

- LCFLAGS= -DIPTABLES_VERSION=\”$(iptables_version)\”

  Becomes

  LCFLAGS= -DIPTABLES_VERSION=\”<IPtables version>\”

  Where <IPtables version> is determined by the statement:

  iptables -V

  An example of the IPtables version is 1.3.5, so <IPtables version> will equal to '1.3.5' without the quotes.

6. Save the Makefile file, and exit the editor.

7. Another adjustment that must be made if errors should occur, is to edit the libpit_osf.c file, and modify the following line under the 'static struct iptables_match osf_match={' section.

   .version=IPTABLES_VERSION,

   Becomes

   .version=”1.3.5”,

   The above changes must include the period, comma and quotation marks.

8. Run the command 'make', which will generate the ipt_osf.ko kernel module.

9. Run the command 'make lib', which will build the libipt_osf.so shared library.

10. Copy the OSF shared library to the directory that IPtables is found in. This is done with the statement:

    ```
    cp ./libipt_osf.so <directory of iptables>
    ```

    Where <directory of iptables> is found with the command:

    ```
    which iptables
    ```

11. Run 'make bin', which will build the application to load fingerprints and obtain information relative to matched packets. These applications are:

    - load – This allows one to load the fingerprints database into the module.

      OSF has the 'load' tool with it allowing one to load and flushthe fingerprint table. It has also the following options availablewith it:

      - --log x

        This logs the operating system even if it doesn't match the desired one. The variables for x are:

  - 0 : log all matched and unknown entries.
  - 1 : log only the first matching entry.
  - 2 : log all matched entries.

- --ttl x

  This option determines if the OS fingerprinting should rely on the TTL field of the IP stack. The options available are:

  - 0 : This is the true IP and TTL comparison, and mostly works for LANs.
  - 1 : Checks if the TTL is less than the fingerprint one, which works for global addresses.
  - 2 : Do not compare TTL at all. This option was developed when Nmap's TTL value was variable. It allows one to detect Nmap, but can produce false results.

- --netlink

  This option allows OSF to log events through netlink connector.

- --connector

  This option is only valid for Linux kernel version 2.6.14 and above, and will also log all events through netlink connector.

- osfd – This is the netlink deamon that listens for incoming matched packets over the netlink.

- ucon_osf - This is the netlink deamon that listens for incoming matched packets using the connector module. This is not being used for the investigations done so far.

12. Run the following command to install the OSF module:

```
insmod ./ipt_osf.ko
```

13. Load the OSF database into the kernel with the following statement:

```
./load ./pf.os /proc/sys/net/ipv4/osf
```

The Nmap packets are blocked with the statement:

```
iptables -I INPUT -p tcp -m osf --genre Nmap --log 2  --ttl 2 -j DROP
```

Should the error:

```
iptables:match 'osf' v(I'm v1.3.5).
```

occur, then it means that the OSF database has not been updated. To correct this repeat step 10.

In case a new OS fingerprinting tool becomes available and OSF is not able to drop the packets from this tool, then the new tools fingerprint can be included in the OSF database. To do this, edit the pf.os file and add the new fingerprint in the format:

<Window Size>:<Initial TTL>:<Don't Fragment bit>:<Overall SYN packet size>:   <Options in Order if used>:   OS Fingerprinting Tool's Name

Repeat step 13 again after the new fingerprint has been added.

The script that is written to load the OSF module and to block Nmap packets has to have steps 12 and 13 in it, as well as the IPtables statement to drop the packets. It is advised that the IPtables statement is placed at the end of the table, to allow communication with Microsoft Windows system, or any other OS that might have a similar fingerprint as that of the OS fingerprinting tool.

Appendix E – Summary of Nmap Options

This summary is from the Nmap manual pages.

Syntax: nmap [Scan Type] [Option] <host of network #1, ... #N>

Target Specifications

-iL <targets_filename>	Input from list
-iR <numhosts>	Choose random targets. *numhosts* specifies the number of IPs that Nmap must generate.
--exclude <host1,host2...>	Exclude these hosts from scanning
--excludefile <excludefile>	Excludes hosts in the specified file from being scanned. The hosts are separated from a TAB delimiter, or by a new line.

Host Discovery

-sL	List / DNS scan
-sP	Ping scan
-P0	Don't ping
-PS [portlist]	SYN ping. E.g. -PS22,23,25
-PA [portlist]	TCP Acknowledge ping
-PU [portlist]	UDP ping
-PE	Echo Ping
-PP	Timestamp ping

-PM	Address Mask ping
-PR	ARP ping
-n	No DNS resolution
-R	Do DNS resolution
--system_dns	Use system DNS resolver (Used for IPv6 scans)
--dns_servers <server1, server2 ...>	Servers to do reverse DNS queries

Port Scanning

-sS	SYN scan
-sU	UDP scan
-sT	TCP connect
-sN	Null scan
-sA	ACK scan
-sF	FIN scan
-sW	Windows scan
-sX	Xmas scan (FIN,PSH and URG flags are set)
-sM	TCP Maimon scan
--scanflags	Custom TCP scan e.g. URGACKPSHRSTSYNFIN will set all the control flags
-sI <zombiehost:port>	Idle scan
-sO	Protocol scan
-b <ftp relay host>	FTP bounce

Port Specifications and Scan Order

-p <port_ranges>	Port ranges to scan
-F	Fast scan mode
-r	don't randomise port scan

Services and Version Detection

-sV	Version detection
--allports	Don't exclude any ports from version detection
--version-intensity <intensity>	Set the intensity of the version detection
--version-light	Enable light mode
--version-all	Try every single port
--version-trace	Trace version scan activity

Operating System (OS) Detection

-O	Operating system detection
--osscan-limit	Limit OS detection to promising targets
--osscan-guess	Guess from the OS detection results
--fuzzy	Same as –osscan-guess

Timing and Performance

--min-hostgroup <numhosts>	Adjust minimum parallel group size
--max-hostgroup <numhosts>	Adjust maximum parallel group size

--min-parallelism <numhosts>	Adjust minimum probe parallelization
--max-parallelism <numhosts>	Adjust maximum probe parallelization
--min_rtt_timeout <time>	Adjust minimum probe timeouts (time is in milliseconds)
--max_rtt_timeout <time>	Adjust maximum probe timeouts
--initial_rtt_timeout <time>	Adjust initial probe timeouts
--max-retries <numtries>	specify the maximum number of portscan probe transmissions
--host-timeout <time>	Give up on slow target hosts
--scan-delay <time>	Delay between probes
--max_scan-delay <time>	Adjust maximum delay between probes
-T Paranoid	Serial scan and 300 seconds wait
-T Sneaky	Serial scan and 15 seconds wait
-T Polite	Serial scan and 0.4 second wait
-T Normal	Parallel scan
-T Aggressive	Parallel scan and 300 seconds timeout and 1.25 seconds per probe
-T Insane	Parallel scan and 75 seconds timeout and 0.3 seconds per probe

Firewall / IDS Evasion and Spoofing

-f	Fragmentation
--mtu	Using the specified MTU
-D <decoy1, decoy2, ME, ...>	Cloak a scan with decoys
-S <IP_Address>	Spoof source address

-e <interface>	Use specified interface
--source-port <portnumber>	Spoof source port number
-g <portnumber>	Same as –source-port <portnumber>
--data-length <number>	Append random data to send packets
--ttl <value>	Set IP time-to-live field
--randomize_hosts	Randomize hosts
--spoof-mac <macinfo>	Spoof MAC address (macinfo can be MAC address, prefix or vendor name)
--badsum	Send packet with bogus TCP/UDP checksums

Output

-oN <filename>	Normal output to filename
-oX <filename>	XML output to filename
-oS <filename>	Script Kiddie Output to filename
-oG <filename>	Grepable output to filename
-oA <basename>	Output to all formats with the basename.*
-v	Increase verbosity level
-d [level]	increases or sets debugging level
--packet-trace	Trace packet and data sent and received
--iflist	List interfaces and routers
--append-output	Append to output file instead of overwriting it
--resume <filename>	Resume aborted scan
--stylesheet <path or URL>	Set XSL stylesheet to transform XML

	output
--webxml	load stylesheet from insecure.org
--no_stylesheet	omit XSL stylesheet declaration from XML

Miscellaneous Options

-6	Enable IPv6 scanning
-A	Aggressive scan options
--datadir <directoryname>	Specify custom Nmap data file location
--send_eth	use raw ethernet sending
--send_ip	send at raw IP level
--privileged	Assume that the user is fully privileged
--interactive	Start in interactive mode
-V	Print version number
--version	Same as -V
-h	Print help summary page
--help	Same as -h

Runtime Interaction

The following keys can be used to change running conditions when Nmap is running:

v / V	Increase / Decrease the verbosity
d / D	Increase / Decrease the debugging level

p / P	Turn on / off packet tracing
?	Print a runtime interaction help screen
Anything else	Prints out status messages

Appendix F – Summary of Xprobe version 2's Options

This is a summary of Xprobe2's manual pages.

Syntax: xprobe2 [Option] <host>

Options

-v	be verbose
-r	display route to target (an output similar to traceroute, displaying the route that the packet travelled)
-p <proto:portnum:state>	Specify the port number, protocol and state. E.g. tcp:23:open; UDP:53:CLOSED . portnum are values between 1 and 65535.
-c <configfile>	Use <configfile> to read the xprobe2.conf configuration file from a different location.
-h	Prints the help of Xprobe
-o <fname>	Log everything to the file <fname>. The default output is to stderr.
-t <time_sec>	Set the round trip or initial receive time-out time. The default time is 10 seconds.
-s <send_delay>	Set the delay between the sending of the packets. <send_delay> is in terms of milliseconds.
-d <debuglv>	Specify the debugging level.

-D <modnum>	Disable the module number <modnum>
-M <modnum>	Enable the module number <modnum>
-L	Display the available modules
-m <numofmatches>	Display the number of matches to print.
-T <portspec>	Enable TCP port scan for specified port(s). E.g. -T21-23,53,110
-U <portspec>	Enable UDP port scan for specified port(s). E.g. -U23-23,53,110
-f	Force fixed round-trip time (-t option)
-F	Generate signature. Use the -o option to save to a file.
-X	Generate XML output and save it to <logfile> specified with the -o option.
-B	Forces TCP handshake module to be able to guess open TCP ports.
-A	Analyse the packets gathered during the port scanning, to be able to detect suspicious traffic, such as transparent proxies and firewalls. This option is to be used with the -T option.

Appendix G – Information analysed by Ethereal

The following information is given by Ethereal on sending or receiving a TCP packet:

Frame number sent or received and it's size

Arrival Time: Date and Time

Time delta from previous packet

Time since reference or first frame

Frame Number

Packet Length

Protocols in frame (e.g. eth:ip:tcp)

Ethernet, Source and Destination Address

Destination

Source

Type: (e.g. IP)

Internet Protocol, Source Address and Destination Address

Version: (e.g. 4)

Header length: (e.g. 20)

Differentiated Services Field: (e.g. Default or ECN)

0000 00 . . = Differentiated Services Codepoint

. . . . 0 . = ENC-Capable Transport (ECT)

. . . . 0 = ECN-CE

Total Length : 40

Identification

Flags

* . . . = Reserved bit (* is either set or unset)

. * . . = Don't fragment (* is either set or unset)

. . * . = More fragments (* is either set or unset)

Fragment offset

Time to live

Protocol: (e.g. TCP)

Header checksum: [Indication whether it is correct or not]

Source Address

Destination Address

Transmission Control Protocol, Source Port and Destination Port, Sequence Number, Acknowledgement Number and Length

Source port

Destination port

Sequence number (relative sequence number)

Header Length

Flags (* is either set or unset)

* . . . . = Congestion Window Reduced (CWR)

. * . . . . . . = ECN-Echo

. . * . . . . . = Urgent

. . . * . . . . = Acknowledgement

. . . . * . . . = Push

. . . . . . * . = Syn

. . . . * = Fin

Window size

Checksum: [Indication whether it is correct or not]

Appendix H – Abstracts of Ethereal with the Xprobe2 tests performed under IPv4

With Firewall Disabled

```
No.     Time        Source                Destination           Protocol Info
      4 6.400527    10.0.0.5              10.0.0.6              ICMP     Echo (ping) request

Frame 4 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.166447000
    Time delta from previous packet: 0.000017000 seconds
    Time since reference or first frame: 6.400527000 seconds
    Frame Number: 4
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x8191 (33169)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                                      (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xe50d [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x22b1 [correct]
    Identifier: 0x8191
    Sequence number: 0x0000
    Data (56 bytes)

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.
0010  00 54 81 91 00 00 40 01 e5 0d 0a 00 00 05 0a 00   .T....@.........
0020  00 06 08 00 22 b1 81 91 00 00 44 dc a3 04 00 02   ....".....D.....
0030  80 d7 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      5 6.400668    10.0.0.6              10.0.0.5              ICMP     Echo (ping) reply

Frame 5 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.166588000
    Time delta from previous packet: 0.000141000 seconds
    Time since reference or first frame: 6.400668000 seconds
    Frame Number: 5
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0de1 (3553)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x58be [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x2ab1 [correct]
    Identifier: 0x8191
    Sequence number: 0x0000                             (Sequence number = 0)
    Data (56 bytes)

0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 00   ...s.%.0.fz...E.
0010  00 54 0d e1 00 00 40 01 58 be 0a 00 00 06 0a 00   .T....@.X.......
0020  00 05 00 00 2a b1 81 91 00 00 44 dc a3 04 00 02   ....*.....D.....
0030  80 d7 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      6 6.434253    10.0.0.5              10.0.0.6              ICMP     Echo (ping) request

Frame 6 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.200173000
    Time delta from previous packet: 0.033585000 seconds
    Time since reference or first frame: 6.434253000 seconds
    Frame Number: 6
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x06 (DSCP 0x01: Unknown DSCP; ECN: 0x02)
        0000 01.. = Differentiated Services Codepoint: Unknown (0x01)
        .... ..1. = ECN-Capable Transport (ECT): 1
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x634d (25421)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                    (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xc34b [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 123
    Checksum: 0x9567 [correct]
    Identifier: 0x8191
    Sequence number: 0x0001
    Data (56 bytes)

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 06   .0.fz....s.%..E.
0010  00 54 63 4d 40 00 40 01 c3 4b 0a 00 00 05 0a 00   .TcM@.@..K......
0020  00 06 08 7b 95 67 81 91 00 01 44 dc a3 04 00 03   ...{.g....D.....
0030  0d a4 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      7 6.434444    10.0.0.6              10.0.0.5              ICMP     Echo (ping) reply

Frame 7 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.200364000
    Time delta from previous packet: 0.000191000 seconds
    Time since reference or first frame: 6.434444000 seconds
    Frame Number: 7
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x06 (DSCP 0x01: Unknown DSCP; ECN: 0x02)
        0000 01.. = Differentiated Services Codepoint: Unknown (0x01)
        .... ..1. = ECN-Capable Transport (ECT): 1
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0de2 (3554)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x58b7 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 123
    Checksum: 0x9d67 [correct]
    Identifier: 0x8191
    Sequence number: 0x0001                    (Sequence number = 1 => incremented)
    Data (56 bytes)

0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 06   ...s.%.0.fz...E.
0010  00 54 0d e2 00 00 40 01 58 b7 0a 00 00 06 0a 00   .T....@.X.......
0020  00 05 00 7b 9d 67 81 91 00 01 44 dc a3 04 00 03   ...{.g....D.....
0030  0d a4 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      8 6.460577    10.0.0.5              10.0.0.6              ICMP     Timestamp request

Frame 8 (54 bytes on wire, 54 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.226497000
    Time delta from previous packet: 0.026133000 seconds
    Time since reference or first frame: 6.460577000 seconds
    Frame Number: 8
    Packet Length: 54 bytes
    Capture Length: 54 bytes
    Protocols in frame: eth:ip:icmp
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0x8191 (33169)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                                    (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xe539 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 13 (Timestamp request)
    Code: 0
    Checksum: 0xfce2 [correct]
    Identifier: 0x8191
    Sequence number: 0x0000
    Originate timestamp: 226440
    Receive timestamp: 0
    Transmit timestamp: 0

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.
0010  00 28 81 91 00 00 40 01 e5 39 0a 00 00 05 0a 00   .(....@..9......
0020  00 06 0d 00 fc e2 81 91 00 00 00 03 74 88 00 00   ............t...
0030  00 00 00 00 00 00                                 ......

No.     Time        Source                Destination           Protocol Info
      9 6.460749    10.0.0.6              10.0.0.5              ICMP     Timestamp reply

Frame 9 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.226669000
    Time delta from previous packet: 0.000172000 seconds
    Time since reference or first frame: 6.460749000 seconds
    Frame Number: 9
    Packet Length: 60 bytes
    Capture Length: 60 bytes
    Protocols in frame: eth:ip:icmp
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
    Trailer: 000000000000
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0x0de3 (3555)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x58e8 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Internet Control Message Protocol
    Type: 14 (Timestamp reply)
    Code: 0
    Checksum: 0x1464 [correct]
    Identifier: 0x8191
    Sequence number: 0x0000
    Originate timestamp: 226440
    Receive timestamp: 56651871
    Transmit timestamp: 56651871

0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 00   ...s.%.0.fz...E.
0010  00 28 0d e3 00 00 40 01 58 e8 0a 00 00 06 0a 00   .(....@.X.......
0020  00 05 0e 00 14 64 81 91 00 00 00 03 74 88 03 60   .....d......t..`
0030  70 5f 03 60 70 5f 00 00 00 00 00 00               p_.`p_......

No.     Time        Source                Destination           Protocol Info
     10 6.484581    10.0.0.5              10.0.0.6              ICMP     Address mask request

Frame 10 (46 bytes on wire, 46 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:20.250501000
    Time delta from previous packet: 0.023832000 seconds
    Time since reference or first frame: 6.484581000 seconds
    Frame Number: 10
    Packet Length: 46 bytes
    Capture Length: 46 bytes
    Protocols in frame: eth:ip:icmp
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 32
    Identification: 0x8191 (33169)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                        (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xe541 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 17 (Address mask request)
    Code: 0
    Checksum: 0x6d6e [correct]
    Identifier: 0x8191
    Sequence number: 0x0000
    Address mask: 0.0.0.0 (0x00000000)

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.
0010  00 20 81 91 00 00 40 01 e5 41 0a 00 00 05 0a 00   . ....@..A......
0020  00 06 11 00 6d 6e 81 91 00 00 00 00 00 00         ....mn........

No.     Time        Source                Destination           Protocol Info
     12 7.270227    10.0.0.6              10.0.0.5              ICMP     Destination unreachable (Port unreachable)

Frame 12 (146 bytes on wire, 146 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:21.036147000
    Time delta from previous packet: 0.000187000 seconds
    Time since reference or first frame: 7.270227000 seconds
    Frame Number: 12
    Packet Length: 146 bytes
    Capture Length: 146 bytes
    Protocols in frame: eth:ip:icmp:ip:udp:dns
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00)
        1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 132
    Identification: 0x0de4 (3556)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                        (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x57cb [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x116d [correct]
    Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
        Version: 4

        Header length: 20 bytes

        Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

            0000 00.. = Differentiated Services Codepoint: Default (0x00)

            .... ..0. = ECN-Capable Transport (ECT): 0

            .... ...0 = ECN-CE: 0

        Total Length: 104

        Identification: 0x0001 (1)

        Flags: 0x04 (Don't Fragment)

            0... = Reserved bit: Not set

            .1.. = Don't fragment: Set

            ..0. = More fragments: Not set

        Fragment offset: 0

        Time to live: 255

        Protocol: UDP (0x11)

        Header checksum: 0x6779 [correct]

            Good: True

            Bad : False

        Source: 10.0.0.5 (10.0.0.5)

        Destination: 10.0.0.6 (10.0.0.6)

    User Datagram Protocol, Src Port: domain (53), Dst Port: 65534 (65534)

        Source port: domain (53)

        Destination port: 65534 (65534)

        Length: 84

        Checksum: 0x5010 [correct]

    Domain Name System (response)

        Transaction ID: 0xbcfd

        Flags: 0x81b0 (Standard query response, No error)

            1... .... .... .... = Response: Message is a response

            .000 0... .... .... = Opcode: Standard query (0)

            .... .0.. .... .... = Authoritative: Server is not an authority for domain

            .... ..0. .... .... = Truncated: Message is not truncated

            .... ...1 .... .... = Recursion desired: Do query recursively

            .... .... 1... .... = Recursion available: Server can do recursive queries
            .... .... .0.. .... = Z: reserved (0)
            .... .... ..1. .... = Answer authenticated: Answer/authority portion was authenticated by the server
            .... .... .... 0000 = Reply code: No error (0)
        Questions: 1
        Answer RRs: 1
        Authority RRs: 0
        Additional RRs: 0
        Queries
            www.securityfocus.com: type A, class IN
                Name: www.securityfocus.com
                Type: A (Host address)
                Class: IN (0x0001)
        Answers
            www.securityfocus.com: type A, class IN, addr 205.206.231.10
                Name: www.securityfocus.com
                Type: A (Host address)
                Class: IN (0x0001)
                Time to live: -12 hours, -56 minutes, -12 seconds
                Data length: 1024
                Addr: 205.206.231.10

    0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 c0   ...s.%.0.fz...E.
    0010  00 84 0d e4 00 00 40 01 57 cb 0a 00 00 06 0a 00   ......@.W.......
    0020  00 05 03 03 11 6d 00 00 00 00 45 00 00 68 00 01   .....m....E..h..
    0030  40 00 ff 11 67 79 0a 00 00 05 0a 00 00 06 00 35   @...gy.........5
    0040  ff fe 00 54 50 10 bc fd 81 b0 00 01 00 01 00 00   ...TP...........
    0050  00 00 03 77 77 77 0d 73 65 63 75 72 69 74 79 66   ...www.securityf
    0060  6f 63 75 73 03 63 6f 6d 00 00 01 00 01 03 77 77   ocus.com......ww
    0070  77 0d 73 65 63 75 72 69 74 79 66 6f 63 75 73 03   w.securityfocus.
    0080  63 6f 6d 00 00 01 00 01 ff ff 4a 14 04 00 cd ce   com.......J.....
    0090  e7 0a                                             ..

    No.     Time        Source                Destination           Protocol Info
         13 7.301929    10.0.0.5              10.0.0.6              TCP      17026 > 65535 [SYN] Seq=0 Ack=0
```

```
Win=6840 Len=0


Frame 13 (54 bytes on wire, 54 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:21.067849000
    Time delta from previous packet: 0.031702000 seconds
    Time since reference or first frame: 7.301929000 seconds
    Frame Number: 13
    Packet Length: 54 bytes
    Capture Length: 54 bytes
    Protocols in frame: eth:ip:tcp
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
        0001 00.. = Differentiated Services Codepoint: Unknown (0x04)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0xe116 (57622)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                            (TTL value of 64)
    Protocol: TCP (0x06)
    Header checksum: 0x459f [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Transmission Control Protocol, Src Port: 17026 (17026), Dst Port: 65535 (65535), Seq: 0, Ack: 0, Len: 0
    Source port: 17026 (17026)
    Destination port: 65535 (65535)
    Sequence number: 0    (relative sequence number)
    Header length: 20 bytes
    Flags: 0x0002 (SYN)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...0 .... = Acknowledgment: Not set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..1. = Syn: Set
        .... ...0 = Fin: Not set
    Window size: 6840                                      (Window size = 6840)
    Checksum: 0x2071 [correct]

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 10   .0.fz....s.%..E.
0010  00 28 e1 16 40 00 40 06 45 9f 0a 00 00 05 0a 00   .(..@.@.E.......
0020  00 06 42 82 ff ff 40 3e dd ee 00 00 00 00 50 02   ..B...@>......P.
0030  1a b8 20 71 00 00                                 .. q..

No.     Time        Source                Destination           Protocol Info
     14 7.302124    10.0.0.6              10.0.0.5              TCP      65535 > 17026 [RST, ACK] Seq=0 Ack=0
Win=0 Len=0

Frame 14 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:21.068044000
    Time delta from previous packet: 0.000195000 seconds
    Time since reference or first frame: 7.302124000 seconds
    Frame Number: 14
    Packet Length: 60 bytes
    Capture Length: 60 bytes
    Protocols in frame: eth:ip:tcp
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
    Trailer: 000000000000
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
        0001 00.. = Differentiated Services Codepoint: Unknown (0x04)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                                    (TTL value of 64)
    Protocol: TCP (0x06)
    Header checksum: 0x26b6 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Transmission Control Protocol, Src Port: 65535 (65535), Dst Port: 17026 (17026), Seq: 0, Ack: 0, Len: 0
    Source port: 65535 (65535)
    Destination port: 17026 (17026)
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 0    (relative ack number)
    Header length: 20 bytes
    Flags: 0x0014 (RST, ACK)
```

```
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .1.. = Reset: Set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 0                                                    (Window size = 0)
    Checksum: 0x3b16 [correct]
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 13
        The RTT to ACK the segment was: 0.000195000 seconds

0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 10   ...s.%.0.fz...E.
0010  00 28 00 00 40 00 40 06 26 b6 0a 00 00 06 0a 00   .(..@.@.&.......
0020  00 05 ff ff 42 82 00 00 00 00 40 3e dd ef 50 14   ....B.....@>..P.
0030  00 00 3b 16 00 00 00 00 00 00 00 00               ..;.........

No.     Time        Source                Destination           Protocol Info
     15 7.302204    10.0.0.5              10.0.0.6              TCP      24472 > 65535 [SYN] Seq=0 Ack=0
Win=6840 Len=0

Frame 15 (54 bytes on wire, 54 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:21.068124000
    Time delta from previous packet: 0.000080000 seconds
    Time since reference or first frame: 7.302204000 seconds
    Frame Number: 15
    Packet Length: 54 bytes
    Capture Length: 54 bytes
    Protocols in frame: eth:ip:tcp
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
        0001 00.. = Differentiated Services Codepoint: Unknown (0x04)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0xd837 (55351)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: TCP (0x06)
    Header checksum: 0x4e7e [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Transmission Control Protocol, Src Port: 24472 (24472), Dst Port: 65535 (65535), Seq: 0, Ack: 0, Len: 0
    Source port: 24472 (24472)
    Destination port: 65535 (65535)
    Sequence number: 0    (relative sequence number)
    Header length: 20 bytes
    Flags: 0x0002 (SYN)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...0 .... = Acknowledgment: Not set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..1. = Syn: Set
        .... ...0 = Fin: Not set
    Window size: 6840                                    (Window size = 6840)
    Checksum: 0xb6c8 [correct]

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 10   .0.fz....s.%..E.
0010  00 28 d8 37 40 00 40 06 4e 7e 0a 00 00 05 0a 00   .(.7@.@.N~......
0020  00 06 5f 98 ff ff 16 c6 53 f9 00 00 00 00 50 02   .._.....S.....P.
0030  1a b8 b6 c8 00 00                                 ......

No.     Time        Source                Destination           Protocol Info
     16 7.302302    10.0.0.6              10.0.0.5              TCP      65535 > 24472 [RST, ACK] Seq=0 Ack=0
Win=0 Len=0

Frame 16 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Aug 11, 2006 17:32:21.068222000
    Time delta from previous packet: 0.000098000 seconds
    Time since reference or first frame: 7.302302000 seconds
    Frame Number: 16
    Packet Length: 60 bytes
    Capture Length: 60 bytes
    Protocols in frame: eth:ip:tcp
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
    Trailer: 000000000000
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)
        0001 00.. = Differentiated Services Codepoint: Unknown (0x04)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 40
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                                    (TTL value of 64)
    Protocol: TCP (0x06)
    Header checksum: 0x26b6 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Transmission Control Protocol, Src Port: 65535 (65535), Dst Port: 24472 (24472), Seq: 0, Ack: 0, Len: 0
    Source port: 65535 (65535)
    Destination port: 24472 (24472)
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 0    (relative ack number)
    Header length: 20 bytes
    Flags: 0x0014 (RST, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 0... = Push: Not set
        .... .1.. = Reset: Set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 0                                       (Window size = 0)
    Checksum: 0xd16d [correct]
    SEQ/ACK analysis
        This is an ACK to the segment in frame: 15
        The RTT to ACK the segment was: 0.000098000 seconds
```

```
0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 10   ...s.%.0.fz...E.
0010  00 28 00 00 40 00 40 06 26 b6 0a 00 00 06 0a 00   .(..@.@.&.......
0020  00 05 ff ff 5f 98 00 00 00 00 16 c6 53 fa 50 14   ...._.......S.P.
0030  00 00 d1 6d 00 00 00 00 00 00 00 00               ...m........
```

With Firewall Enabled

```
No.     Time        Source                Destination           Protocol Info
      1 0.000000    10.0.0.5              10.0.0.6              ICMP     Echo (ping) request

Frame 1 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:33:33.771341000
    Time delta from previous packet: 0.000000000 seconds
    Time since reference or first frame: 0.000000000 seconds
    Frame Number: 1
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0xaabf (43711)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xbbdf [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb557 [correct]
    Identifier: 0xaabf
    Sequence number: 0x0000
    Data (56 bytes)

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.
0010  00 54 aa bf 00 00 40 01 bb df 0a 00 00 05 0a 00   .T....@.........
0020  00 06 08 00 b5 57 aa bf 00 00 44 dc a3 4d 00 0b   .....W....D..M..
0030  c4 b0 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      2 0.000244    10.0.0.6              10.0.0.5              ICMP     Echo (ping) reply

Frame 2 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:33:33.771585000
    Time delta from previous packet: 0.000244000 seconds
    Time since reference or first frame: 0.000244000 seconds
    Frame Number: 2
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
```

```
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)

    Source: 10.0.0.6 (00:30:18:66:7a:0b)

    Type: IP (0x0800)

Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)

    Version: 4

    Header length: 20 bytes

    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

        0000 00.. = Differentiated Services Codepoint: Default (0x00)

        .... ..0. = ECN-Capable Transport (ECT): 0

        .... ...0 = ECN-CE: 0

    Total Length: 84

    Identification: 0x0de5 (3557)

    Flags: 0x00

        0... = Reserved bit: Not set

        .0.. = Don't fragment: Not set

        ..0. = More fragments: Not set

    Fragment offset: 0

    Time to live: 64                                                  (TTL value of 64)

    Protocol: ICMP (0x01)

    Header checksum: 0x58ba [correct]

        Good: True

        Bad : False

    Source: 10.0.0.6 (10.0.0.6)

    Destination: 10.0.0.5 (10.0.0.5)

Internet Control Message Protocol

    Type: 0 (Echo (ping) reply)

    Code: 0

    Checksum: 0xbd57 [correct]

    Identifier: 0xaabf

    Sequence number: 0x0000

    Data (56 bytes)


0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 00   ...s.%.0.fz...E.

0010  00 54 0d e5 00 00 40 01 58 ba 0a 00 00 06 0a 00   .T....@.X.......
```

```
0020  00 05 00 00 bd 57 aa bf 00 00 44 dc a3 4d 00 0b   .....W....D..M..
0030  c4 b0 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   ................
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                 67

No.     Time        Source                Destination           Protocol Info
      3 0.030562    10.0.0.5              10.0.0.6              ICMP     Echo (ping) request

Frame 3 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:33:33.801903000
    Time delta from previous packet: 0.030318000 seconds
    Time since reference or first frame: 0.030562000 seconds
    Frame Number: 3
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x06 (DSCP 0x01: Unknown DSCP; ECN: 0x02)
        0000 01.. = Differentiated Services Codepoint: Unknown (0x01)
        .... ..1. = ECN-Capable Transport (ECT): 1
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0ead (3757)
    Flags: 0x04 (Don't Fragment)
        0... = Reserved bit: Not set
        .1.. = Don't fragment: Set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                          (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x17ec [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 123
    Checksum: 0x3d66 [correct]
    Identifier: 0xaabf
    Sequence number: 0x0001
    Data (56 bytes)

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 06   .0.fz....s.%..E.
0010  00 54 0e ad 40 00 40 01 17 ec 0a 00 00 05 0a 00   .T..@.@.........
0020  00 06 08 7b 3d 66 aa bf 00 01 44 dc a3 4d 00 0c   ...{=f....D..M..
0030  3c 25 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   <%..............
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37                                             67

No.     Time        Source                Destination           Protocol Info
      4 0.030759    10.0.0.6              10.0.0.5              ICMP     Echo (ping) reply

Frame 4 (98 bytes on wire, 98 bytes captured)
    Arrival Time: Aug 11, 2006 17:33:33.802100000
    Time delta from previous packet: 0.000197000 seconds
    Time since reference or first frame: 0.030759000 seconds
    Frame Number: 4
    Packet Length: 98 bytes
    Capture Length: 98 bytes
    Protocols in frame: eth:ip:icmp:data
Ethernet II, Src: 10.0.0.6 (00:30:18:66:7a:0b), Dst: 10.0.0.5 (00:0e:a6:73:e7:25)
    Destination: 10.0.0.5 (00:0e:a6:73:e7:25)
    Source: 10.0.0.6 (00:30:18:66:7a:0b)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.6 (10.0.0.6), Dst: 10.0.0.5 (10.0.0.5)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x06 (DSCP 0x01: Unknown DSCP; ECN: 0x02)
        0000 01.. = Differentiated Services Codepoint: Unknown (0x01)
        .... ..1. = ECN-Capable Transport (ECT): 1
        .... ...0 = ECN-CE: 0
    Total Length: 84
    Identification: 0x0de6 (3558)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                          (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x58b3 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.6 (10.0.0.6)
    Destination: 10.0.0.5 (10.0.0.5)
Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 123
    Checksum: 0x4566 [correct]
    Identifier: 0xaabf
    Sequence number: 0x0001
    Data (56 bytes)
```

```
0000  00 0e a6 73 e7 25 00 30 18 66 7a 0b 08 00 45 06   ...s.%.0.fz...E.

0010  00 54 0d e6 00 00 40 01 58 b3 0a 00 00 06 0a 00   .T....@.X.......

0020  00 05 00 7b 45 66 aa bf 00 01 44 dc a3 4d 00 0c   ...{Ef....D..M..

0030  3c 25 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15   <%..............

0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25   .......... !"#$%

0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35   &'()*+,-./012345

0060  36 37                                       67


No.     Time        Source                Destination           Protocol Info

      5 0.063754    10.0.0.5              10.0.0.6              ICMP     Timestamp request


Frame 5 (54 bytes on wire, 54 bytes captured)

    Arrival Time: Aug 11, 2006 17:33:33.835095000

    Time delta from previous packet: 0.032995000 seconds

    Time since reference or first frame: 0.063754000 seconds

    Frame Number: 5

    Packet Length: 54 bytes

    Capture Length: 54 bytes

    Protocols in frame: eth:ip:icmp

Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)

    Destination: 10.0.0.6 (00:30:18:66:7a:0b)

    Source: 10.0.0.5 (00:0e:a6:73:e7:25)

    Type: IP (0x0800)

Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)

    Version: 4

    Header length: 20 bytes

    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

        0000 00.. = Differentiated Services Codepoint: Default (0x00)

        .... ..0. = ECN-Capable Transport (ECT): 0

        .... ...0 = ECN-CE: 0

    Total Length: 40

    Identification: 0xaabf (43711)

    Flags: 0x00

        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                              (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0xbc0b [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 13 (Timestamp request)
    Code: 0
    Checksum: 0x8a57 [correct]
    Identifier: 0xaabf
    Sequence number: 0x0000
    Originate timestamp: 835036
    Receive timestamp: 0
    Transmit timestamp: 0

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.
0010  00 28 aa bf 00 00 40 01 bc 0b 0a 00 00 05 0a 00   .(....@.........
0020  00 06 0d 00 8a 57 aa bf 00 00 00 0c bd dc 00 00   .....W..........
0030  00 00 00 00 00 00                                 ......

No.     Time        Source                Destination           Protocol Info
      6 0.252004    10.0.0.5              10.0.0.6              ICMP     Address mask request

Frame 6 (46 bytes on wire, 46 bytes captured)
    Arrival Time: Aug 11, 2006 17:33:34.023345000
    Time delta from previous packet: 0.188250000 seconds
    Time since reference or first frame: 0.252004000 seconds
    Frame Number: 6
    Packet Length: 46 bytes
    Capture Length: 46 bytes
    Protocols in frame: eth:ip:icmp
Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)
    Destination: 10.0.0.6 (00:30:18:66:7a:0b)
    Source: 10.0.0.5 (00:0e:a6:73:e7:25)
    Type: IP (0x0800)
Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..0. = ECN-Capable Transport (ECT): 0
        .... ...0 = ECN-CE: 0
    Total Length: 32
    Identification: 0xcfbe (53182)
    Flags: 0x00
        0... = Reserved bit: Not set
        .0.. = Don't fragment: Not set
        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                                    (TTL value of 64)
    Protocol: ICMP (0x01)
    Header checksum: 0x9714 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Internet Control Message Protocol
    Type: 17 (Address mask request)
    Code: 0
    Checksum: 0x1f41 [correct]
    Identifier: 0xcfbe
    Sequence number: 0x0000
    Address mask: 0.0.0.0 (0x00000000)
```

```
0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 00   .0.fz....s.%..E.

0010  00 20 cf be 00 00 40 01 97 14 0a 00 00 05 0a 00   . ....@.........

0020  00 06 11 00 1f 41 cf be 00 00 00 00 00 00         .....A........


No.     Time        Source                Destination           Protocol Info

      8 2.258017    10.0.0.5              10.0.0.6              TCP      27624 > 65535 [SYN] Seq=0 Ack=0
Win=6840 Len=0


Frame 8 (54 bytes on wire, 54 bytes captured)

    Arrival Time: Aug 11, 2006 17:33:36.029358000

    Time delta from previous packet: 0.999683000 seconds

    Time since reference or first frame: 2.258017000 seconds

    Frame Number: 8

    Packet Length: 54 bytes

    Capture Length: 54 bytes

    Protocols in frame: eth:ip:tcp

Ethernet II, Src: 10.0.0.5 (00:0e:a6:73:e7:25), Dst: 10.0.0.6 (00:30:18:66:7a:0b)

    Destination: 10.0.0.6 (00:30:18:66:7a:0b)

    Source: 10.0.0.5 (00:0e:a6:73:e7:25)

    Type: IP (0x0800)

Internet Protocol, Src: 10.0.0.5 (10.0.0.5), Dst: 10.0.0.6 (10.0.0.6)

    Version: 4

    Header length: 20 bytes

    Differentiated Services Field: 0x10 (DSCP 0x04: Unknown DSCP; ECN: 0x00)

        0001 00.. = Differentiated Services Codepoint: Unknown (0x04)

        .... ..0. = ECN-Capable Transport (ECT): 0

        .... ...0 = ECN-CE: 0

    Total Length: 40

    Identification: 0x303f (12351)

    Flags: 0x04 (Don't Fragment)

        0... = Reserved bit: Not set

        .1.. = Don't fragment: Set

        ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 64                                          (TTL value of 64)
    Protocol: TCP (0x06)
    Header checksum: 0xf676 [correct]
        Good: True
        Bad : False
    Source: 10.0.0.5 (10.0.0.5)
    Destination: 10.0.0.6 (10.0.0.6)
Transmission Control Protocol, Src Port: 27624 (27624), Dst Port: 65535 (65535), Seq: 0, Ack: 0, Len: 0
    Source port: 27624 (27624)
    Destination port: 65535 (65535)
    Sequence number: 0    (relative sequence number)
    Header length: 20 bytes
    Flags: 0x0002 (SYN)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...0 .... = Acknowledgment: Not set
        .... 0... = Push: Not set
        .... .0.. = Reset: Not set
        .... ..1. = Syn: Set
        .... ...0 = Fin: Not set
    Window size: 6840                                   (Window size = 6840)
    Checksum: 0x9621 [correct]

0000  00 30 18 66 7a 0b 00 0e a6 73 e7 25 08 00 45 10   .0.fz....s.%..E.
0010  00 28 30 3f 40 00 40 06 f6 76 0a 00 00 05 0a 00   .(0?@.@..v......
0020  00 06 6b e8 ff ff 3e 13 41 03 00 00 00 00 50 02   ..k...>.A.....P.
0030  1a b8 96 21 00 00                                 ...!..
```

Appendix I – GNU General Public License

GNU GENERAL PUBLIC LICENSE

****

Version 2, June 1991

Preamble

========

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if