ICMP Usage in Scanning

The Complete Know-How

Ofir Arkin

Founder

The Sys-Security Group

http://www.sys-security.com
ofir@sys-security.com

Version 3.0

June 2001

Trust No One

Table of Contents

Figures List

Table List

Diagram List

1.0 Introduction

1.1 Introduction to Version 1.0

The ICMP Protocol may seem harmless at first glance. Its goals and features were outlined in RFC 792 (and than later cleared in RFCs 1122, 1256, 1349, 1812), as a way to provide a means to send error messages for non-transient error conditions, and to provide a way to probe the network in order to determine general characteristics about the network. In terms of security, ICMP is one of the most controversial protocols in the TCP/IP protocol suite. The risks involved in implementing the ICMP protocol in a network, regarding scanning, are the subject of this research paper.

Scanning will usually be the major stage of an information gathering process a malicious computer attacker will launch against a targeted network. With this stage the malicious computer attacker will try to determine what are the characteristics of the targeted network. He will use several techniques, such as host detection, service detection, network topology mapping, and operating system fingerprinting. The data collected will be used to identify those Hosts (if any) that are running a network service, which may have a known vulnerability. This vulnerability may allow the malicious computer attacker to execute a remote exploit in order to gain unauthorized access to those systems. This unauthorized access may become his focal point to the whole targeted network.

This research paper outlines the usage of the ICMP protocol in the scanning process. Step-by-Step we will uncover each of the malicious computer attacker techniques using the ICMP protocol. A few new scanning techniques will be unveiled in this research paper. I have reported some of them to several security mailing lists, including Bugtraq, in the past.

The chapters in this research paper are divided according to the various scanning techniques:

- Host Detection using the ICMP protocol is dealt in Chapter 3.
- Advanced Host Detection methods using the ICMP protocol are discussed in Chapter 4.
- Inverse mapping using the ICMP protocol is discussed in Chapter 5.
- Network mapping using the `traceroute` utility and other tools is discussed in Chapter 6.
- Chapter 7 discusses the usage of ICMP in the Active Operating System Fingerprinting process.
- Chapter 9 suggests a filtering policy to be used on filtering devices when dealing with the ICMP protocol.

I would like to take a stand in this controversial issue. ICMP protocol hazards are not widely known. I hope this research paper will change this fact.

1.2 Introduction to Version 2.0

Quite a large number of new operating system fingerprinting methods using the ICMP protocol, which I have found are introduced with this revision. Among these methods two can be used in order to identify Microsoft Windows 2000 machines; one will allow us to distinguish between Microsoft based operating system based machines and the rest of the world, and another will allow us to distinguish between Sun Solaris machines and the rest of the world. I have also tried to be accurate as possible with data presented in this paper. Few tables have been added to the paper mapping the behavior of the various operating systems I have used. These tables describe the results I got from the various machines after querying them with the various tests introduced with this paper.

See section 1.3 for a full Changes list.

1.3 Introduction to Version 2.5

With this version of the research paper I am introducing a few new OS fingerprinting methods. Some are targeted in producing ICMP error messages from a target OS, enabling us to fingerprint an OS even if all ports of the OS in question are closed. I have also added a considerable amount of information about ICMP error message. At the end of the paper you will find the Basic snort rule base I have written.

1.4 Introduction to Version 3.0

The work for Version 3.0 has started when I have built the Training Session for the Black Hat (http://www.blackhat.com) Windows 2000 Security conference. I felt that a more ordered paper is needed. I have decided to start the paper with a full explanation (including examples) of the ICMP protocol. I have felt that the research paper will be easier to understand once you have read the overview about the ICMP protocol.

I have introduced a new section with this version dealing with “Passive Fingerprinting Using the ICMP Protocol” (chapter 8).

Snort rules were written to deal with all of the examples and methods given in this paper.

Some new active operating system fingerprinting methods were explained with this version.

2.0 The ICMP Protocol

The **I**nternet **C**ontrol **M**essage **P**rotocol goals and features were outlined in RFC 792 as a way to provide a means to send error messages for non-transient error conditions, and to provide a way to probe the network in order to determine general characteristics about the network.

RFC 1122 (Requirements for Internet Hosts – Communication Layers), and RFC 1812 (Requirements for IP version 4 Routers) later clarified some of the ICMP protocol features.

In order to work reliably and consistently with other implementations of the ICMP protocol, we need to incorporate RFC 792, RFC 1122, and RFC 1812.

Other RFCs have defined other functionalities for the ICMP protocol:

- RFC 896 - Source Quench
- RFC 950 - Address Mask Extensions
- RFC 1191 - Path MTU Discovery
- RFC 1256 - Router Discovery
- RFC 1349 - Type of Service in the Internet Protocol Suite[1]

A more accurate definition of the Internet Control Message Protocol goals and features might be that it is used for two types of operations:

- When a *router* or a *destination host* need to inform the source host about errors in a datagram processing, and

- For probing the network with request & reply messages in order to determine general characteristics about the network.

2.1 The ICMP Specifications

ICMP messages are sent in IP datagrams. Although ICMP uses IP as if it were a higher-level protocol, ICMP is an internal part of IP, and must be implemented in every IP module.

It is important to note that the ICMP protocol is used to provide feedback about some errors (non-transient) in a datagram processing, not to make IP reliable. Datagrams may still be undelivered without any report of their loss. If a higher level protocol that use IP need reliability he must implement it.

RFC 792 defines the IP protocol ID for ICMP to be 1. It also states that the IP Type-of-Service field value and the Precedence Bits value should be equal to zero. According to RFC 1812, Routers will use the value of 6 or 7 as their IP Precedence bits value with ICMP Error messages.

2.1.1 Special Conditions with ICMP messages

For transient error messages no ICMP error message should be sent. For the following conditions the ICMP protocol has strict rules of inner working which are defined in RFC 792:

[1] Now being replaced by the DiffServ mechanism. For more information refer to RFC 2474 (http://www.ietf.org/rfc/rfc2474.txt).

- No ICMP Error messages are sent **in response to ICMP Error messages** to avoid infinite repetition[2].
- For fragmented IP datagrams ICMP messages are **only sent for errors on fragment zero** (the first fragment).
- ICMP Error messages are never sent in response to a datagram that is **destined to a broadcast or a multicast address**.
- ICMP Error messages are never sent in response to a datagram sent as a **link layer broadcast**.
- ICMP Error messages are never sent in response to a datagram whose **source address does not represents a unique host** – the source IP address cannot be *zero*, a *loopback address*, a *broadcast address* or a *multicast address*.
- ICMP Error messages are never sent in response to an **IGMP message** of any kind.
- When an ICMP message of **unknown type** is received, it must be silently discarded.
- Routers will almost always generate ICMP messages but when it comes to a destination host(s), the number of ICMP messages generated is implementation dependent.

From a closer look at the various rules, we can conclude that a thought about a “network storm”, and extra network traffic were behind most of the ICMP protocol special conditions.

2.2 ICMP Messages

The ICMP protocol is used for two types of operations:

- Reporting non-transient error conditions.
- Probing the network with request & reply messages in order to determine general characteristics about the network.

A number code, also known as the “message type”, is assigned to each ICMP message; it specifies the type of the message.

Another number code represents a “code” for the specified ICMP type. It acts as a sub-type, and its interpretation is dependent upon the message type.

The ICMP protocol has two types of operations; therefore its messages are also divided to two:

- ICMP Error Messages
- ICMP Query Messages

The **I**nternet **A**ssigned **N**umbers **A**uthority (IANA) has a list defining the ICMP message types that are currently registered. It also lists the RFC that defines the ICMP message. The list is available at: http://www.isi.edu/in-notes/iana/assignments/icmp-parameters.

Table 1 defines the various ICMP types and codes.

[2] ICMP Error messages can be sent for ICMP Query messages, when generating a non-transient error condition.

Type	Name	Code
0	Echo Reply	0 No Code
1	Unassigned	
2	Unassigned	
3	Destination Unreachable[3]	0 Net Unreachable 1 Host Unreachable 2 Protocol Unreachable 3 Port Unreachable 4 Fragmentation Needed and Don't Fragment was Set 5 Source Route Failed 6 Destination Network Unknown 7 Destination Host Unknown 8 Source Host Isolated[4] 9 Communication with Destination Network is Administratively Prohibited[5] 10 Communication with Destination Host is Administratively Prohibited[6] 11 Destination Network Unreachable for Type of Service. 12 Destination Host Unreachable for Type of Service. 13 Communication Administratively Prohibited. 14 Host Precedence Violation 15 Precedence cutoff in effect
4	Source Quench	0 No Code
5	Redirect	0 Redirect Datagram for the Network (or subnet) 1 Redirect Datagram for the Host 2 Redirect Datagram for the Type of Service and Network 3 Redirect Datagram for the Type of Service and Host
6	Alternate Host Address	0 Alternate Address for Host
7	Unassigned	
8	Echo Request	0 No Code
9	Router Advertisement	0 No Code
10	Router Selection	0 No Code
11	Time Exceeded	0 Time to Live exceeded in Transit 1 Fragment Reassembly Time Exceeded
12	Parameter Problem	0 Pointer indicates the error 1 Missing a Required Option 2 Bad Length
13	Timestamp	0 No Code
14	Timestamp Reply	0 No Code

[3] RFC 972 defines codes 1-5. RFC 1122 defines codes 6-12. RFC 1812 defines codes 13-15.
[4] Reserved for use by U.S. military agencies.
[5] Reserved for use by U.S. military agencies.
[6] Reserved for use by U.S. military agencies.

Type	Name	Code
15	Information Request	0 No Code
16	Information Reply	0 No Code
17	Address Mask Request	0 No Code
18	Address Mask Reply	0 No Code
19	Reserved (for Security)	0 No Code
	20-29 reserved (for Robustness Experiment)	
30	Traceroute	
31	Datagram Conversion Error	
32	Mobile Host Redirect	
33	IPv6 Where-Are-You	
34	IPv6 I-Am-Here	
35	Mobile Registration Request	
36	Mobile Registration Reply	
39	SKIP	
40	Photuris	
		0 Reserved 1 unknown security parameters index 2 valid security parameters, but authentication failed 3 valid security parameters, but decryption failed

Table 1: ICMP Types & Codes

The ICMP messages differ in structure and formatting because of their different functionality. The general ICMP message format is defined by the next figure:

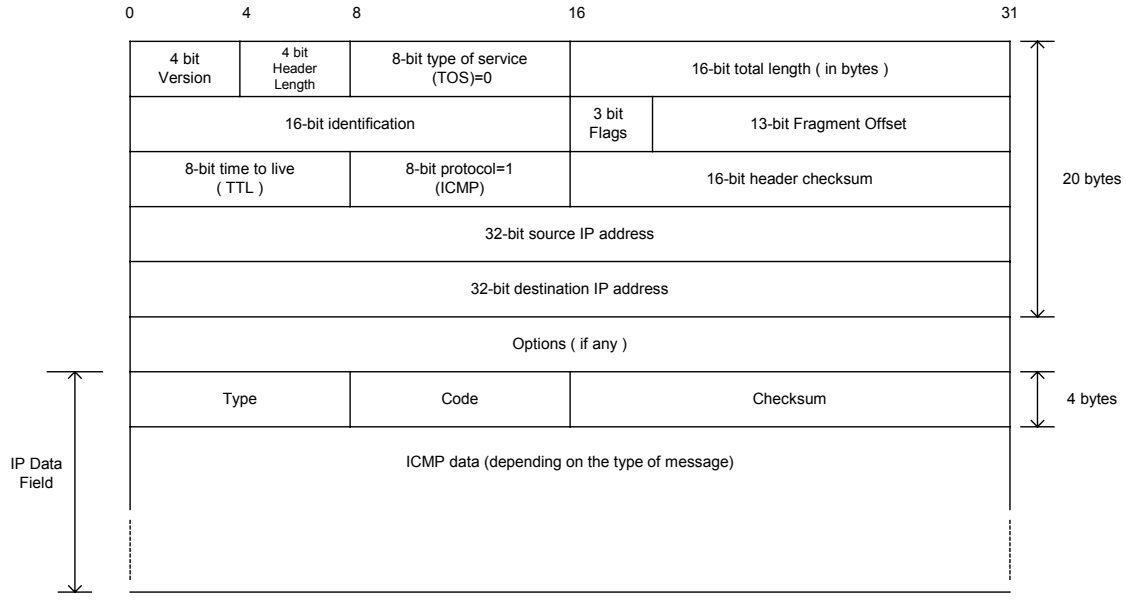


Figure 1: ICMP Message Format

2.2.1 ICMP Error Messages

ICMP error messages are used to report a problem that prevented delivery. The nature of the problem should be a non-transient delivery problem.

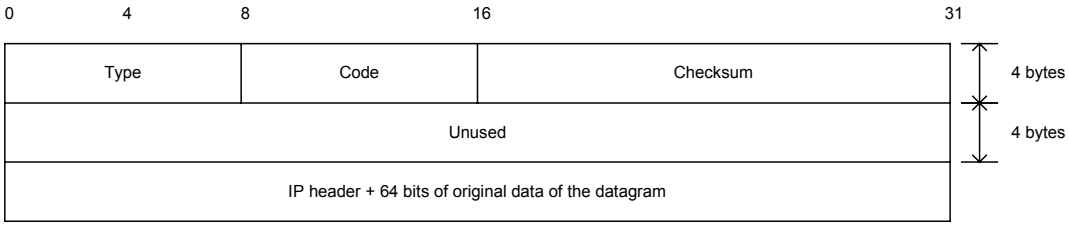


Figure 2: ICMP Error Message General Format

Some fields within the ICMP Error messages are always sent:

Field	Size	Notes
Type	1 byte	Indicate the ICMP error message type
Code	1 byte	Indicate the specific sub-type of the ICMP error message
Checksum	2 bytes	Validation of the ICMP Header
Original IP Header	20-60 bytes	The IP Header of the offending packet.
Original Data	8 bytes[7]	The first 64 bits of the Offending Packet's data.

Table 2: ICMP Error Messages Common Fields

ICMP error message length

Every ICMP error message includes the IP Header (20 to 60 bytes) and *at least* the first 8 data bytes of the datagram that triggered the error; more than 8 bytes *may* be sent; this header and data must be unchanged from the received datagram.

An ICMP error message length should be, therefore, between 36 to 72 bytes[8].

ICMP Error Messages
Destination Unreachable (Type 3)
Source Quench (Type 4)
Redirect (Type 5)
Time Exceeded (Type 11)
Parameter Problem (Type 12)

Table 3: ICMP Error Messages

[7] There might be more than 8 bytes of data from the offending packet.
[8] There might be more than 8 bytes of data from the offending packet being quoted with the ICMP error message. Therefore the datagram size will be bigger than the usual. I will demonstrate this later in the paper.

RFC 792 defines the IP protocol ID for ICMP to be 1. It also states that the IP Type-of-Service field value and the Precedence Bits value should be equal to zero. According to RFC 1812, Routers will use the value of 6 or 7 as their IP Precedence bits value with ICMP Error messages.

The ICMP Protocol Rules for ICMP Error Messages

- ICMP Error messages are never sent for another ICMP Error message to prevent infinite loops.
- ICMP error messages are never sent in response to a datagram that is *destined* to a *broadcast* or a *multicast* address.
- ICMP error messages are never sent in response to a datagram sent as a link layer broadcast.
- ICMP error messages are never sent in response to a datagram whose source address does not represents a unique host – the source IP address cannot be *zer*o, a *loopback* address, a *broadcast* address or a *multicast* address.
- ICMP Error messages are never sent in response to an IGMP massage of any kind.

The conditions for issuing error messages by Routers and Host(s) differ. Therefore I will outline the conditions for issuing the error messages separately.

2.2.1.1 Destination Unreachable (Type 3)

ICMP Destination Unreachable message type issued by a Destination Host:
A *destination host* issues a destination unreachable message when the protocol specified in the *protocol number* field of the original datagram is not active on the destination host, or the *specified port* is inactive.

ICMP Destination Unreachable message type issued by a Router:
A *router* issue a destination unreachable message in response to a packet that it cannot forward because the destination (or next hop) is unreachable or a service is unavailable.

Code	Meaning	Explanation
0	Network Unreachable	Generated by a router if a route to the destination network is not available.
1	Host Unreachable	Generated by a router if a route to the destination host on a directly connected network is not available (does not respond to ARP).
2	Protocol Unreachable	Generated if the transport protocol designated in a datagram is not supported in the transport layer of the final destination.
3	Port Unreachable	Generated if the designated transport protocol (e.g. UDP) is unable to demultiplex the datagram in the transport layer of the final destination but has no protocol mechanism to inform the sender.
4	Fragmentation needed and DF flag Set	Generated if a router needs to fragment but cannot since the DF flag is set.
5	Source Route Failed	Generated if a router cannot forward a packet to the next hop in a source route option.

Code	Meaning	Explanation
6	Destination Network Unknown	According to RFC 1812 this code should not be generated since it would imply on the part of the router that the destination network does not exist (net unreachable code 0 should be used instead of code 6).
7	Destination Host Unknown	Generated only when a router can determine (from link layer advice) that the destination host does not exist.
8	Source Host Isolated	Generated by a Router if it have been configured not to forward packets from source.
9	Communication with Destination Network is Administratively Prohibited	Generated by a Router if it has been configured to block access to the desired destination network.
10	Communication with Destination Host is Administratively Prohibited	Generated by a Router if it has been configured to block access to the desired destination host.
11	Network Unreachable for Type of Service	Generated by a router if a route to the destination network with the requested or default TOS is not available.
12	Host Unreachable for Type of Service	Generated if a router cannot forward a packet because its route(s) to the destination do not match either the TOS requested in the datagram or the default TOS (0).
13*	Communication Administratively Prohibited	Generated if a router cannot forward a packet due to administrative filtering (ICMP sender is not available at this time).
14	Host Precedence Violation	Sent by the first hop router to a host to indicate that a requested precedence is not permitted for the particular combination of source/destination host or network, upper layer protocol, and source/destination port.
15	Precedence cutoff in effect	The network operators have imposed a minimum level of precedence required for operation, the datagram was sent with precedence below this level.

***** Routers *may* have a configuration option that causes code 13 messages not to be generated. When this option is enabled, no ICMP error message is sent in response to a packet that is dropped because it's forwarding is administratively prohibited. Same is with type 14 & 15.

Table 4: Destination Unreachable Codes (Router)

2.2.1.1.1 Destination Unreachable – Fragmentation Needed but the Don't Fragment Bit was set

The only type of ICMP Destination Unreachable error message, which is slightly different in its datagram format from the other destination unreachable ICMP error messages format, is Type 3 Code 4 – Fragmentation Needed but the Don't Fragment Bit was set.

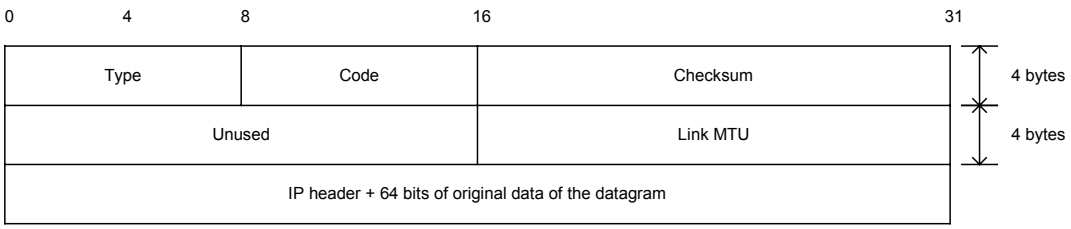


Figure 3: ICMP Fragmentation Needed but the Don't Fragment Bit was set Message Format

The Unused field will be 16 bits in length, instead of 32 bits, with this type of message. The rest of the 16 bits will be used to carry the MTU (Maximum Transfer Unit) used for the link that could not deliver the datagram to the next hop (or destination) because the size of the datagram was too big to carry. Since this datagram could not be fragmented (the DF Bit was set) an error message has been sent to the sender indicating that a lower MTU should be used, hinting the size of the next hops links.

2.2.1.1.2 Destination Unreachable - Communication with Destination Network is Administratively Prohibited

The Error message indicates that the destination system is configured to reject datagrams from the sending system. This error is used when datagrams based on some sort of criteria are being filtered by a filtering device (firewall/router/other filtering devices) restrictions or other security measures.

We can conclude that our Destination Host is up and running, but we cannot reach it, since the filtering device is blocking our packets, and is instructing us to stop sending datagrams.

With the next example a router is configured to block all requests, coming from the Internet, targeting port 53 on the destination machine it applies its ACL on:

```
05/09/01-12:29:41.399543 RoutersIP -> SourceIP
ICMP TTL:244 TOS:0x0 ID:24442 IpLen:20 DgmLen:56
Type:3  Code:13  DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
SourceIP:4667 -> DestinationIP:53
TCP TTL:53 TOS:0x0 ID:40019 IpLen:20 DgmLen:60
**U****F Seq: 0x97EABAF6  Ack: 0x1C1D1E1F  Win: 0x2223  TcpLen: 8
UrgPtr: 0x2627
** END OF DUMP
00 00 00 00 45 00 00 3C 9C 53 40 00 35 06 29 B0  ....E..<.S@.5.).
xx xx xx xx yy yy yy yy 12 3B 00 35 97 EA BA F6  .....Z...;.5....
```

2.2.1.2 Source Quench (Type 4)

ICMP Source Quench message type issued by a Router:

If a *router* sends this message, it means that the router does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network.

RFC 1812 specify that a router *should not* generate Source Quench messages, but a router that does originate Source Quench message *must* be able to limit the rate at which they are generated (because it consumes bandwidth and it is an ineffective antidote to congestion).

A router receiving an ICMP Source Quench message type:
When a router receives such a message it *may* ignore it.

ICMP Source Quench message type issued by a Host:
If a *destination host* sends this message (it *may* be implemented), it means that the datagrams arrive too fast to be processed. The ICMP source quench message is a request to the host to cut back the rate, which it is sending traffic to the Internet destination.

The ICMP header code would be always zero.

With the next example an HPUX B.11.0 based machine issued an ICMP Source Quench error message:

```
10:48:43.197728 eth0 < 172.18.2.5 > 172.18.2.201: icmp: source quench
Offending pkt: 172.18.2.201 > 172.18.2.5: icmp: echo reply (DF) (ttl
255, id 0) (DF) (ttl 255, id 43363)
                         4500 0070 a963 4000 ff01 7536 ac12 0205
                         ac12 02c9 0400 fbff 0000 0000 4500 0054
                         0000 4000 ff01 1eb6 ac12 02c9 ac12 0205
                         0000 67dc 0761 081f 3b0b 4f4b 0006 fe46
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

Host receiving an ICMP Source Quench message type:
An ICMP Source Quench message *must* be reported to the transport layer, UDP or TCP, the host should throttle itself back for a period of time, than gradually increase the transmission rate again.

2.2.1.3 Redirect (Type 5)

ICMP Redirect message type issued by a Router:
If a router generates this message, it means the host should send future datagrams for the network to the router who's IP is given in the ICMP message. *The router should be always on the same subnet as the host who sent the datagram and the router that generated the ICMP redirect message*.

A routing loop is generated when the router IP address matches the source IP address in the original datagram header.

Routers *must not* generate a Redirect Message unless *all* the following conditions are met:

- The packet is being forwarded out the same physical interface that it was received from,
- The IP source address in the packet is on the same Logical IP (Sub) network as the next-hop IP address, and

- The packet does not contain an IP source route option.

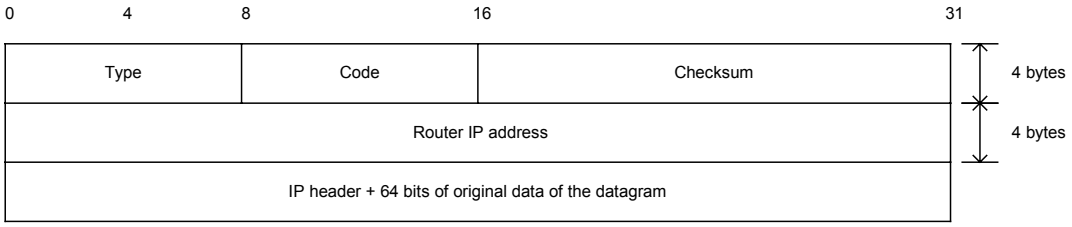


Figure 4: ICMP Redirect Message Format

A router receiving an ICMP Redirect message type:

A router *may* ignore ICMP Redirects when choosing a path for a packet originated by the router if the router is running a routing protocol or if forwarding is enabled on the router and on the interface over which the packet is being sent.

Four different codes can appear in the code field:

Code	Meaning
0	Redirect Datagram for the Network (or subnet)
1	Redirect Datagram for the Host
2	Redirect Datagram for the Type of Service and Network
3	Redirect Datagram for the Type of Service and Host

Table 5: Redirect Codes

ICMP Redirect message type issued by a Host:

A *host should not* send an ICMP Redirect message. Redirects are to be sent only by routers[9].

Host receiving an ICMP Redirect message type:

A *host* receiving a Redirect message *must* update its routing information accordingly. Every host *must* be prepared to accept both Host and Network Redirects.

The Redirect message *should* be silently discarded with the following cases:

- The new gateway address it specifies is not on the same connected (sub-) net through which the Redirect arrived.
- If the source of the Redirect is not the current first-hop gateway for the specified destination.

[9] A Router cannot differentiate between an ICMP Redirect coming from a Router, and between an ICMP Redirect coming from a Host. This is infect a good example of relying upon OS implementation to be according to the RFC guideline.

2.2.1.4 Time Exceeded (Type 11)

ICMP Time-To-Live Exceeded in Transit Error message issued by a Router:

The sending operating system (or application) sets the time to live field in the IP header to a value that represents the maximum time the datagram is allowed to travel on the Internet.

The field value is decreased at each point that the Internet header (IP Header) is being processed. RFC 791 states that this field decreasement reflects the time spent processing the datagram. The field value is measured in units of seconds. The RFC also states that the maximum time to live value can be set to 255 seconds, which equals 4.25 minutes. The datagram must be discarded if this field value equals zero - before reaching its destination.

Relating to this field as a measure to assess time is a bit misleading. Some routers may process the datagram faster than a second, and some may process the datagram longer than a second (heavy load situations).

The real intention is to have an upper bound to the datagram's lifetime, so infinite loops of undelivered datagrams will not jam the Internet.

Having a bound to the datagram's lifetime help us to prevent old duplicates to arrive after a certain time elapsed. So when we retransmit a piece of information which was not previously delivered we can be assured that the older duplicate is already discarded and will not interfere with the process.

If a *router* discovers that the Time-To-Live field in an IP header of a datagram he process equals zero he will discard the datagram and generate an ICMP Time Exceeded Code 0 – Time-To-Live Exceeded in Transit (this can also be an indicator of a routing loop problem).

A router *must* generate an ICMP Time Exceeded message code 0 when it discards a packet due to an expired TTL field. A router *may* have a per-interface option to disable origination of these messages on that interface, but that option *must* default to allowing the messages to be originated.

In the next example, after an attempt to 'ping' a certain IP (y.y.y.y), we received an ICMP Time-to-Live Exceeded in transit error message from a Router in route to the destination IP address. The Time-to-Live field value has been expired:

```
05/13/01-16:05:47.639747 RouterIP -> 172.18.2.201
ICMP TTL:117 TOS:0x0 ID:61586 IpLen:20 DgmLen:56
Type:11  Code:0  TTL EXCEEDED
00 00 00 00 45 00 00 54 00 00 40 00 01 01 FA 0F  ....E..T..@.....
AC 12 02 C9 yy yy yy yy 08 00 F1 67 4F 1B 01 00  .....Z.d...gO...
```

ICMP Fragment Reassembly Time Exceeded Error message issued by a Router:

When the router reassembles a packet that is destined for the router, it is acting as an Internet host. Host rules apply also when the router *receives* a Time Exceeded message.

ICMP Time Exceeded message type issued by a Host:

When does an IP fragmentation occur?

- On the sending host - When an application or a transport layer protocol request to send more data than a single IP datagram the underlying network can carry.
- A Router along the path to the destination - When packets move from a network with a higher MTU onto a network with a small MTU.

Each fragment is being transported by a different packet. Therefore each fragment will be routed independently. All fragments will share a common IP identification value in the IP header (helping the reassembly process). Each fragment will carry a unique byte offset value helping to place its carried data in the correct order when reassembly occurs. Except for the last fragment, each fragment will set the MF bit (more fragments) so the receiving host will understand that there are more fragments coming.

The entire datagram must be completely reassembled by the receiving host before it will be handed off to higher levels of the protocol stack.

If a *host* cannot reassemble a fragmented datagram due to missing fragments within its time limit it will discard the datagram and generate an ICMP Time Exceeded Code 1 – Fragment Reassembly Time Exceeded.

Code	Meaning
0	Time-To-Live Exceeded in Transit
1	Fragment Reassembly Time Exceeded

Table 6: time Exceeded Codes

2.2.1.5 Parameter Problem (Type 12)

ICMP Parameter Problem message is sent when a router (*must* generate this message) or a host (*should* generate this message) process a datagram and finds a problem with the IP header parameters. It is only sent if the error caused the datagram to be discarded.

The ICMP Parameter Problem error message is generated usually for any error in the IP header not specifically covered by another ICMP message.

If code 0 is used, the pointer field will point to the exact byte in the original IP Header, which caused the problem (see figure 5).

Three different codes can appear in the code field:

Codes	Meaning	Explanation
0	Pointer indicates the error (unspecified error)	There is a specific problem with the datagram. The pointer indicates the location of the problem.

Codes	Meaning	Explanation
1	Missing a Required Option	The required IP option has not been defined. This message is used by the U.S. Military when using Security options.
2	Bad Length	The Header Length and/or The Total Packet Length values of the IP datagram are not accurate.

Table 7: Parameter Problem Codes

Receipt of a parameter problem message generally indicates some local or remote implementation error.

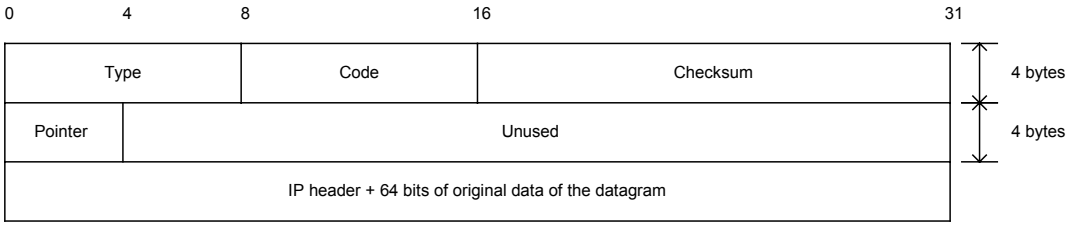


Figure 5: ICMP Parameter Problem Message Format

2.2.2 ICMP Query Messages

ICMP Query messages are being used for probing the network with request & reply messages in order to determine general characteristics about the network. The general characteristics can range from host availability to network latency.

All ICMP Query messages share some characteristics that are summarized in the figure below:

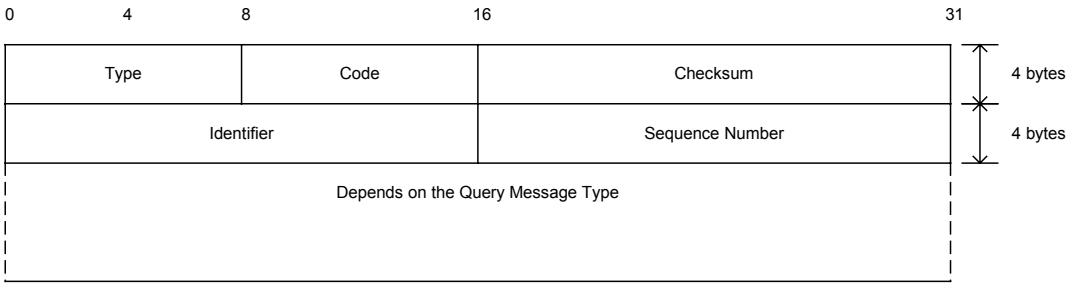


Figure 6: ICMP Query Message Format

The type, code and checksum fields are common to all ICMP message types.

The identifier field is used to differentiate between ICMP query messages sent to different hosts. When initiating an ICMP query request each host receives its own identifier field value.

The sequence number field is used to differentiate between the ICMP query messages sent to the same host.

The fields following are dependent upon the ICMP query message type.

Field	Size	Notes
Type	1 byte	Indicate the ICMP query message type
Code	1 byte	Indicate the specific sub-type of the ICMP query message
Checksum	2 bytes	Validation of the ICMP Header
Identifier	2 Bytes	Used to differentiate between ICMP query messages sent to different hosts. When initiating an ICMP query request each host receives its own identifier field value.
Sequence Number	2 Bytes	Used to differentiate between the ICMP query messages sent to the same host.
Data / Additional Fields	Variable	The fields following are dependent upon the ICMP query message type.

Table 8: ICMP Query Messages – Common Fields

The Length of an ICMP query message type varies from one query message type to another. The ICMP Header will be always 4 bytes. The size of the ICMP Identifier field and the size of the ICMP Sequence Number field will always be the same as well. The only variable in our equation is the additional field's length (that will vary from one ICMP query message type to another).

RFC 792 defines the IP protocol ID for ICMP to be 1. RFC 1122 states that the IP Type-of-Service field value and the Precedence Bits value should be equal to zero. It also states that if a user wishes to set these fields to a different value, than the response (the reply) must use the same IP Type-of-Service and Precedence Bits values, which were used with the ICMP query message.

ICMP Query Messages[10]
ECHO Request (Type 8), and Reply (Type 0)
Time Stamp Request (Type 13), and Reply (Type 14)
Information Request (Type 15), and Reply (Type 16)
Address Mask Request (Type 17), and Reply (Type 18)

Table 9: The ICMP Query Messages

[10] Router Solicitation (Type 10), and Router Advertisement (Type 9) is also considered to be an ICMP Query message type.

The only ICMP query message type, which is common with all operating systems, is the ICMP Echo request. RFC 1122 states that every host should implement an end-user-accessible application interface for sending ICMP Echo request query messages to other hosts.

2.2.2.1 Echo Request (Type 8) and Echo Reply (Type 0)

We can use an *ICMP Echo* datagram to determine whether a target IP address is active or not, by simply sending an ICMP Echo (ICMP type 8) datagram to the targeted system and waiting to see if an *ICMP Echo Reply* (ICMP type 0) is received. If an ICMP Echo reply is received, it would indicate that the target is alive; No response means the target is down.

From a technical point of view: The sending side initializes the identifier (used to identify Echo requests aimed at different destination hosts) and sequence number (if multiple Echo requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP Echo to the destination host. In the ICMP header the code equals zero. The recipient should *only change* the type to Echo Reply, recalculate the ICMP header Checksum, and return the datagram to the sender.

The data received in the Echo message must be returned in the Echo Reply message unchanged.

This mechanism is used by the Ping utility to determine if a destination host is reachable.

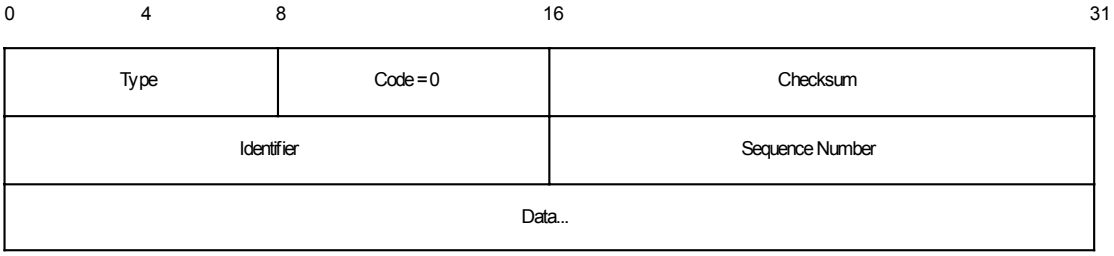


Figure 7: ICMP ECHO Request & Reply message format

The expected behavior from a router/host when handling an ICMP Echo type message is[11]:

- A *router should* have a configuration option that, if enabled, causes the router to silently ignore all ICMP Echo requests; if provided, this option *must* be default to allowing responses.
- Every *host/router* must implement an ICMP Echo server function that receives Echo requests and sends corresponding Echo Replies.
- A host/router *should* implement an application-layer interface for sending an Echo request and receiving an Echo reply, for diagnostic purposes.
- If we send an ICMP Echo request to an IP Broadcast or IP Multicast address it *may* be silently discarded by a host/router.

[11] RFC 1122 requirements for Internet Hosts (http://www.ietf.org/rfc/rfc1122.txt) -- Communication Layers. RFC 1812 Requirements for IP version 4 Routers (http://www.ietf.org/rfc/rfc1812.txt).

- If a Record Route and/or Timestamp option is received in an ICMP Echo request, this option (these options) *should* be updated to include to current router/destination host and included in the IP header of the Echo Reply message, without truncation. Thus, the record route will be for the entire round trip.

- If a Source Route option is received in an ICMP Echo request, the return route *must* be reversed and used as a source route option for the Echo Reply message. A router will not perform this if it is aware of a policy that would prevent the delivery of the message.

ICMP Echo request data size

The amount of data used in the data field within the ICMP Echo request will vary from one implementation to another (and between one family of operating systems to another).

The 'ping' utility with UNIX and UNIX-like operating systems will use an ICMP data field of 56 bytes, adding that to the 20 bytes of the IP header and to the ICMP header (8 bytes) will result in a datagram size of 84 bytes.

The 'ping' utility with Microsoft Windows operating systems will build, by default, an ICMP Echo request datagram with the size of 60 bytes. This is since the 'ping' utility is using a data field of 32 bytes only.

2.2.2.2 Timestamp Request (Type 13) and Timestamp Reply (Type 14)

The *ICMP Time Stamp Request and Reply* allows a node to query another for the current time. This allows a sender to determine the amount of latency that a particular network is experiencing. The sender initializes the identifier (used to identify Timestamp requests aimed at different destination hosts) and sequence number (if multiple Timestamp requests are sent to the same destination host), sets the originate time stamp and sends it to the recipient.

The receiving host fills in the receive and transmit time stamps, change the type of the message to time stamp reply and returns it to the recipient. The time stamp is the number of milliseconds elapsed since midnight UT (GMT).

The originate time stamp is the time the sender last touched the message before sending it, the receive time stamp is the time the recipient first touched it on receipt, and the Transmit time stamp is the time the receiver last touched the message on sending it.

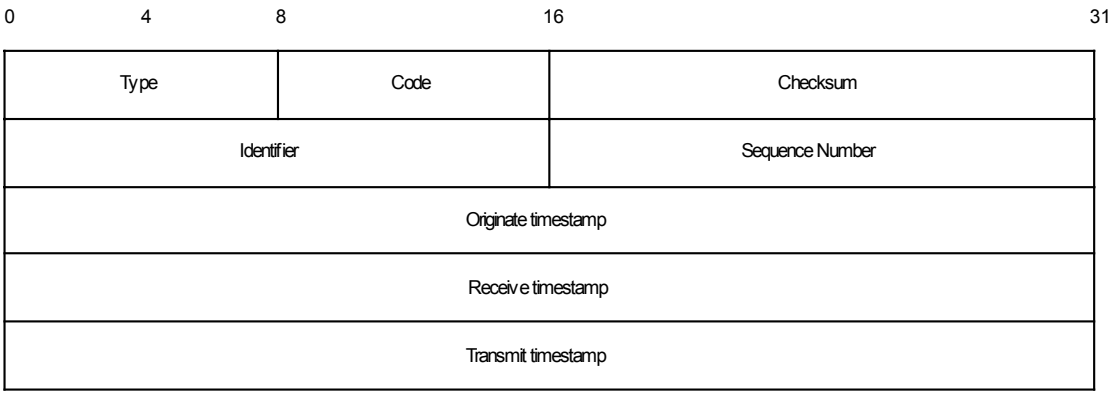


Figure 8: ICMP Time Stamp Request & Reply message format

As RFC 1122 state, a *host/router may* implement Timestamp and Timestamp Reply. If they are implemented a Host/Router must follow these rules:

- Minimum variability delay in handling the Timestamp request.
- The receiving host *must* answer to every Timestamp request that he receives.
- An ICMP Timestamp Request to an IP Broadcast or IP Multicast address *may* be silently discarded.
- The IP source address in an ICMP Timestamp reply *must* be the same as the specific-destination address of the corresponding Timestamp request message.
- If a source-route option is received in a Timestamp request, the return route *must* be reserved and used as a Source Route option for the Timestamp Reply option.
- If a Record Route and/or Timestamp option is received in a Timestamp request, this option(s) *should* be updated to include the current host and included in the IP header of the Timestamp Reply message.

The ICMP Timestamp message should be between 40 to 60 bytes long. Combined from the IP header (20-40 bytes), the ICMP header (4 bytes), and the ICMP Timestamp related fields (16 bytes).

In the next example I have issued an ICMP Timestamp request from a host running Linux Kernel 2.4 (172.18.2.201), to another Linux based host running Linux Kernel 2.2.16 (172.18.2.200)[12].

```
05/13/01-15:58:58.799747 172.18.2.201 -> 172.18.2.200
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:40
Type:13  Code:0  TIMESTAMP REQUEST
FA 04 00 00 02 C9 2D 70 00 00 00 00 00 00 00 00  ......-p........

05/13/01-15:58:58.799747 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:21170 IpLen:20 DgmLen:40
Type:14  Code:0  TIMESTAMP REPLY
FA 04 00 00 02 C9 2D 70 61 91 FF 02 61 91 FF 02  ......-pa...a...
```

2.2.2.3 Information Request (Type 15) and Reply (Type 16)

The *ICMP Information Request/Reply* pair was intended to support self-configuring systems such as diskless workstations at boot time, to allow them to discover their network address.

The sender (a host) fills in the request with the Destination IP address in the IP Header set to zero (meaning this network). The request may be sent with both Source IP Address and Destination IP Address set to zero. The sender initializes the identifier and the sequence number, both used to match the replies with the requests, and sends out the request. The ICMP header code field is zero.

If the request was issued with a non-zero Source IP Address the reply would only contain the network address in the Source IP Address of the reply. If the request had both the Source IP Address and the Destination IP Address set to zero, the reply will contain the network address in both the source and destination fields of the IP header.

[12] I was using the sing utility (http://www.sourceforge.net/projects/sing) to generate the ICMP Timestamp request.

From the description above one can understand that the ICMP Information request and reply mechanism was intended to be used locally.

The RARP, BOOTP & DHCP protocols provide better mechanisms for hosts to discover its own IP address.

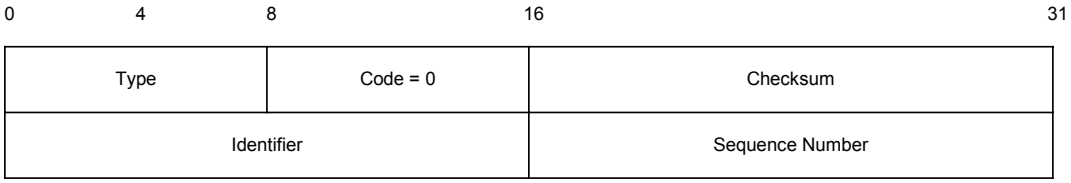


Figure 9: ICMP Information Request & Reply message format

The ICMP Information request & reply messages are combined from the IP header (20-40 bytes), the ICMP header (4 bytes), and the ICMP Identifier and Sequence number fields (4 bytes). Therefore an ICMP Information request or reply message should be between 28 to 48 bytes long.

The Information Request & Reply mechanism is now obsolete as stated in RFC 1122, and RFC 1812[13]. A *router should not* originate or respond to these messages; a *host should not* implement these messages.

2.2.2.4 ICMP Address Mask Request (Type 17) and Reply (Type 18)

The *ICMP Address Mask Request* (and Reply) is intended for diskless systems to obtain its subnet mask in use on the local network at bootstrap time. Address Mask request is also used when a node wants to know the address mask of an interface. The reply (if any) contains the mask of that interface.

Once a host has obtained an IP address, it could than send an Address Mask request message to the broadcast address of the network they reside on (255.255.255.255). Any host on the network that has been configured to send address mask replies will fill in the subnet mask, change the type of the message to address mask reply and return it to the sender.

RFC 1122 states that the Address Mask request & reply query messages are entirely optional.

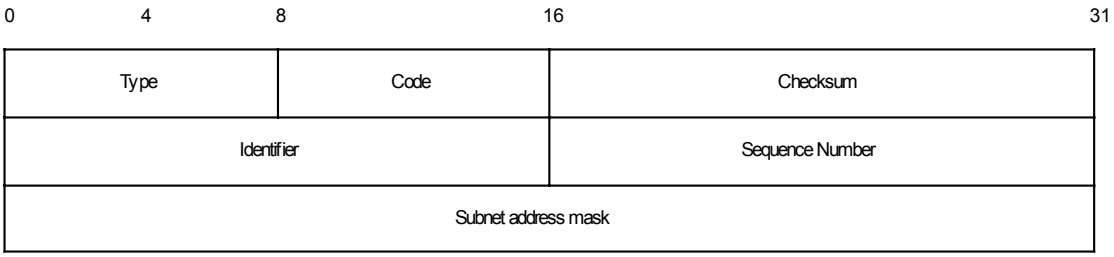


Figure 10: ICMP Address Mask Request & Reply message format

[13] RFC 1812: Requirements for IP Version 4 Routers, http://www.ietf.org/rfc/rfc1812.txt . As the RFC states this mechanism is now obsolete - A *router should not* originate or respond to these messages; A *host should not* implement these messages.

RFC 1122 also states that a system that has implemented ICMP Address Mask messages *must not* send an Address Mask Reply unless it is an authoritative agent for address masks.

Usually an Address Mask request would be answered by a gateway (router or a host acting as a router).

Please note that a Router *must* implement ICMP Address Mask messages. This will help identify routers along the path to the targeted network (it can also reveal internal routers if this kind of traffic is allowed to reach them).

If the Router is following RFC 1812 closely, it should not forward on an Address Mask Request to another network.

An ICMP Address Mask request or reply message is combined from the IP header (20-40 bytes), the ICMP header (4 bytes), and the address mask related fields (8 bytes). Therefore the ICMP address mask request/reply message should be between 32 to 52 bytes long.

Characteristics of Address Mask Request & Reply for a Router:

- A router *must* implement support for receiving ICMP Address Mask Request messages and responding with ICMP Address Mask Reply messages.
- A router *should* have a configuration option for each logical interface specifying whether the router is allowed to answer Address Mask Requests for that interface; this option *must* default to allowing responses.
- A router *must not* respond to an Address Mask Request before the router knows the correct address mask.
- A router *must not* respond to an Address Mask Request that has a source address of 0.0.0.0 and which arrives on a physical interface that has associated with it multiple logical interfaces and the address masks for those interfaces are not all the same.
- A router *should* examine all ICMP Address Mask Replies that it receives to determine whether the information it contains matches the router's knowledge of the address mask. If the ICMP Address Mask Reply appears to be in error, the router *should* log the address mask the sender's IP address. A router *must not* use the contents of an ICMP Address Mask Reply to determine the correct address mask.

Because hosts may not be able to learn the address mask if a router is down when the host boots up, a router *may* broadcast a gratuitous ICMP Address Mask Reply on each of its logical interfaces after it configured its own address masks. However, this feature can be dangerous in environments that use variable length address masks. Therefore, if this feature is implemented, gratuitous Address Mask Replies *must not* be broadcast over any logical interface(s) which either:

- Are not configured to send gratuitous Address Mask Replies. Each logical interface *must* have a configuration parameter controlling this, and that parameter *must* default to not sending the gratuitous Address Mask Replies.
- Share subsuming (but not identical) network prefixes and physical interface.

Characteristics of Address Mask Request & reply for a Host:

A host *must* support the first, and *may* implement all three, of the following methods for determining the address mask(s) corresponding to its IP address(es):

- Static configuration information;

- Obtaining the address mask(s) dynamically as a side effect of the system initialization process; and
- Sending ICMP Address Mask Request(s) and receiving ICMP Address Mask Reply(s).

The choice of method to be used in a particular host *must* be configurable.

When the last method (Sending ICMP Address Mask Request(s) and receiving ICMP Address Mask Reply(s)), the use of Address Mask messages, is enabled, then:

- When it initializes, the host *must* broadcast an Address Mask Request message on the connected network corresponding to the IP address. It *must* retransmit this message a small number of times if it does not receive an immediate Address Mask Reply.

- Until it has received an Address Mask Reply, the host *should* assume a mask appropriate for the address class of the IP address, i.e., assume that the connected network is not subnetted.

- The first Address Mask Reply message received *must* be used to set the address mask corresponding to the particular local IP address. This is true even if the first Address Mask Reply message is "unsolicited", in which case it will have been broadcast and may arrive after the host has ceased to retransmit Address Mask Requests. Once the mask has been set by an Address Mask Reply, later Address Mask Reply messages MUST be (silently) ignored.

Conversely, if Address Mask messages are *disabled*, then no ICMP Address Mask Requests will be sent, *and any ICMP Address Mask Replies received for that local IP address must be (silently) ignored*.

A system *must not* send an Address Mask Reply unless it is an *authoritative agent for address masks*. An authoritative agent may be a host or a gateway, but it *must* be explicitly as an address mask agent. Receiving an address mask via an Address Mask Reply does not give the receiver authority and *must not* be used as the basis for issuing Address Mask Replies.

With a statically configured address mask, there *should* be an additional configuration flag that determines whether the host is to act as an authoritative agent for this mask, i.e., whether it will answer Address Mask Request messages using this mask.

If it is configured as an agent, the host *must* broadcast an Address Mask Reply for the mask on the appropriate interface when it initializes.

2.3 Special Cases - The Path MTU Discovery Process

ICMP "Fragmentation Needed but the Don't Fragment Bit was set" and the Path MTU Discovery Process [14]

[14] RFC 1191, http://www.ietf.org/rfc/rfc1191.txt, J. Mogul, S. Deering.

When one host needs to send data to another host, the data is transmitted in a series of IP datagrams. We wish the datagrams be the largest size possible that does not require fragmentation[15] along the path from the source host to the destination host.

Fragmentation by the IP layer raises few problems:

- If one fragment from a packet is dropped, we need to retransmit the whole packet.
- Load on the routers, which needs to do the fragmentation.
- Some simpler firewalls would block all fragments because they do not contain the header information for a higher layer protocol needed for filtering.

The **M**aximum **T**ransfer **U**nit (***MTU***) is a link layer restriction on the maximum number of bytes of data in a single transmission. The smallest MTU of any link on the current path between two hosts is called the ***Path MTU***.

2.3.1 The PATH MTU Discovery Process

We use the Don't Fragment Bit Flag in the IP header to dynamically discover the Path MTU of a given route. The source host assumes that the PMTU of a path is the known MTU of its first hop. He will send all datagrams with that size, and set the Don't Fragment Bit. If along the path to the destination host, there is a router that needs to fragment the datagram in order to pass it to the next hop, an ICMP error message (Type 3 Code 4 "Fragmentation Needed and DF set") will be generated, since the Don't Fragment bit was set. When the sending host receives the ICMP error message he should reduce his assumed PMTU for the path.

The process can end when the estimated PMTU is low enough for the datagrams not to be fragmented. The source host itself can stop the process if he is willing to have the datagrams fragmented in some circumstances.

Usually the DF bit would be set in all datagrams, so if a route changes to the destination host, and the PMTU is lowered, than we would discover it.

The PMTU of a path might be increased over time, again because of a change in the routing topology. To detect it, a host should periodically increase its assumed PMTU for that link.

The link MTU field in the ICMP "Fragmentation Needed and DF set" error message, carries the MTU of the constricting hop, enabling the source host to know the exact value he needs to set the PMTU for that path to allow the voyage of the datagrams beyond that point (router) without fragmentation.

2.3.2 Host specification

A host must reduce his estimated PMTU for the relevant path when he receives the ICMP "Fragmentation Needed and the DF bit was set" error message. RFC 1191 does not outline a specific behavior that is expected from the sending host, because different applications may have different requirements, and different implementation architectures may favor different strategies.

The only required behavior is that a host *must* attempt to avoid sending more messages with the same PMTU value in the near future. A host can either cease setting the Don't Fragment bit in the

[15] When we send a packet that it is too large to be sent across a link as a single unit, a router needs to slice/split the packet into smaller parts, which contain enough information for the receiver to reassemble them. This is called fragmentation.

IP header (and allow fragmentation by the routers in the way) or reduce the datagram size. The better strategy would be to lower the message datagram size because fragmentation will cause more traffic and consume more Internet resources.

A host using the PMTU Discovery process *must* detect decreases in Path MTU as fast as possible. A host *may* detect increases in Path MTU, by sending datagrams larger than the current estimated PMTU, which will usually be rejected by some router on the path to a destination since the PMTU usually will not increase. Since this would generate traffic back to the host, the check for the increases must be done at infrequent intervals. The RFC specify that an attempt for detecting an increasment *must not* be done less than 10 minutes after a datagram "too big" has been received for the given destination, or less than 2 minute after a previously successful attempt to increase.

The sending host must know how to handle an ICMP "Fragmentation Needed and the DF bit was set" error message that was sent by a device who does not know how to handle the PMTU protocol and does not include the next-hop MTU in the error message. Several strategies are available:

- The PMTU should be set to the minimum between the currently assumed PMTU and 576[16]. The DF bit should not be set in future datagrams for that path.
- Searching for the accurate value for the PMTU for a path. We keep sending datagrams with the DF bit set with lowered PMTU until we do not receive ICMP errors.

A host must not reduce the estimation of a Path MTU value below 68 bytes.

A host must not increase its estimate of the Path MTU in response to the contents of a Datagram Too Big message.

2.3.3 Router Specification

When a router cannot forward a datagram because it exceeded the MTU of the next-hop network and the Don't Fragment bit was set, he is required to generate an ICMP Destination Unreachable message to the source of the datagram., with the appropriate code indicating "Fragmentation needed and the Don't Fragment Bit was set". In the error message the router *must* include the MTU of the next-hop in a 16bit field inside the error message.

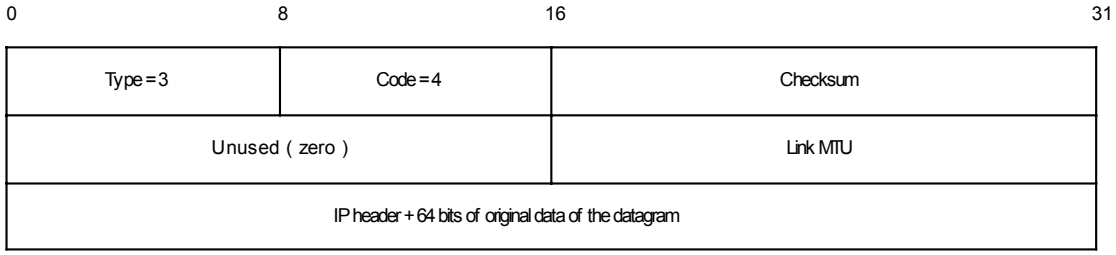


Figure 11: ICMP Fragmentation Required with Link MTU

[16] The usage of the lesser between 576 and the first-hop MTU as the PMTU for a destination, which is not connected to the same network was the old implementation. The results were the use of smaller datagrams than necessary, waste of Internet resources, and not being optimal.

The value of the next-hop MTU field should be set to the size in bytes of the largest datagram that could be forwarded, along the path of the original datagram, without being fragmented by this router. The size includes IP header plus IP data and no lower level headers should be included.

Because every router should be able to forward a datagram of 68 bytes without fragmenting it, the link MTU field should not contain a value less than 68.

2.3.4 The TCP MSS (Maximum Segment Size) Option and PATH MTU Discovery Process

The RFC specify that a host that is doing Path MTU Discovery *must not* send datagrams larger than 576 bytes unless the receiving host grants him permission.

When we are establishing a TCP connection both sides announce the maximum amount of data in one packet that should be sent by the remote system – The maximum segment size, MSS (if one of the ends does not specify an MSS, it defaults to 536 – there is no permission from the other end to send more than this amount). The packet generated would be, normally, 40 bytes larger than the MSS; 20 bytes for the IP header and 20 bytes for the TCP header. Most systems announce an MSS that is determined from the MTU on the interface that the traffic to the remote system passes out from the system through.

Each side upon receiving the MSS of the other side should not send any segments larger than the MSS received, regardless of the PMTU. After receiving the MSS value the Path MTU Discovery process will start to take affect. We will send our IP packets with the DF bit set allowing us to recognize points in the path to our destination that cannot process packets larger as the MSS of the destination host plus 40 bytes. When such an ICMP error message arrives, we should lower the PMTU to a path (according to the link MTU field, or if not used, to use the rules regarding the old implementation) and retransmit. The value of the link MTU cannot be higher than the MSS of the destination host. When retransmission occurs resulting from ICMP type 3 code 4 error message, the congestion windows should not change, but slow start should be initiated. The process continues until we adjust the correct PMTU of a path (not receiving ICMP error messages from the intermediate routers) which will allow us to fragment at the TCP layer which is much more efficient than at the IP layer.

3.0 Host Detection using the ICMP Protocol[17]

The Host Detection stage gives a malicious computer attacker crucial information by identifying the hosts on the targeted network that are reachable from the Internet. This process belongs to the scanning stage, which is one of the first stages in the Information Gathering process. The information collected during this stage could later lead to an attempt to break in to one (or more) of the targeted network computers. This, if the information gathered would be sufficient for the malicious computer attacker.

In this section I will go over basic Host Detection methods using the ICMP protocol. I will also introduce you with several techniques in doing so.

There are no internal OS built tools to generate ICMP query request messages. We will use 3rd party applications/utilities to do so. The OS Kernel implementation of the different ICMP query mechanisms is usually being called by the OS and not triggered by a user. If the ICMP query and reply mechanism is enabled than the OS Kernel will be the one to answer a query. We can examine the Address Mask request and reply mechanism for a good example.

3.1 ICMP Echo (Type 8) and Echo Reply (Type 0)

We can use an *ICMP Echo* datagram to determine whether a target IP address is active or not, by simply sending an ICMP Echo[18] (ICMP type 8) datagram to the targeted system and waiting to see if an *ICMP Echo Reply* (ICMP type 0) is received. If an ICMP Echo reply is received, it would indicate that the target is alive (few firewalls spoof ICMP Echo replies from protected hosts); No response means the target is down or a filtering device is preventing the incoming ICMP Echo datagram from getting inside the protected network or the filtering device prevents the initiated reply from reaching the Internet.

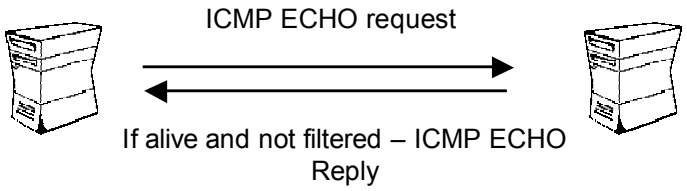


Figure 12: ICMP Echo Mechanism

This mechanism is used by the '`ping`' utility to determine if a destination host is reachable.

In the next example two Linux machines demonstrate the usage of ping. One is based on Kernel 2.4.2 (172.18.2.201), and one is based on Kernel 2.2.16 (172.18.2.200):

```
[root@godfather /root]# ping 172.18.2.200
PING 172.18.2.200 (172.18.2.200) from 172.18.2.201 : 56(84) bytes of
data.
64 bytes from 172.18.2.200: icmp_seq=0 ttl=255 time=617 usec
64 bytes from 172.18.2.200: icmp_seq=1 ttl=255 time=2.489 msec
```

[17] For more information about the ICMP Protocol please refer to Section 2.0: "The ICMP Protocol".

[18] From a technical point of view: The sending side initializes the identifier (used to identify Echo requests aimed at different destination hosts) and sequence number (if multiple Echo requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP Echo to the destination host. In the ICMP header the code equals zero. The recipient should *only change* the type to Echo Reply and return the datagram to the sender.

```
64 bytes from 172.18.2.200: icmp_seq=2 ttl=255 time=2.499 msec
64 bytes from 172.18.2.200: icmp_seq=3 ttl=255 time=2.499 msec

--- 172.18.2.200 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.617/2.026/2.499/0.813 ms
```

The `snort` trace[19]:

```
05/14/01-11:55:30.171542 172.18.2.201 -> 172.18.2.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:58628   Seq:768  ECHO
82 9D FF 3A 5C 9E 02 00 08 09 0A 0B 0C 0D 0E 0F  ...:\...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

05/14/01-11:55:30.171542 172.18.2.200 -> 172.18.2.201
ICMP TTL:255 TOS:0x0 ID:769 IpLen:20 DgmLen:84
Type:0  Code:0  ID:58628  Seq:768  ECHO REPLY
82 9D FF 3A 5C 9E 02 00 08 09 0A 0B 0C 0D 0E 0F  ...:\...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

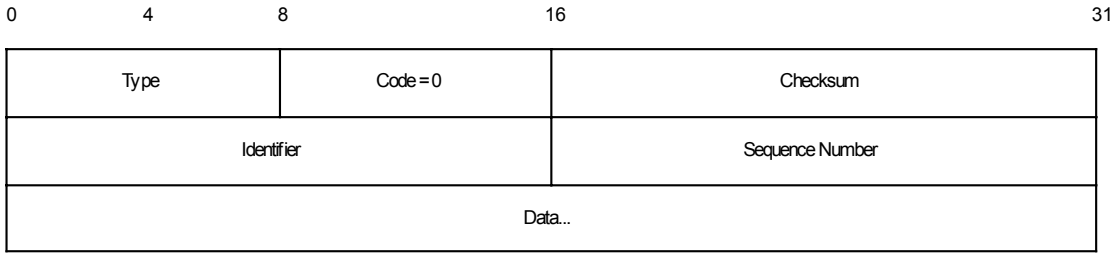


Figure 13: ICMP Echo Request & Reply message format

Countermeasure: Block ICMP Echo requests coming from the Internet towards your network at your border router and/or Firewall[20]. You can also configure your host(s) not to answer ICMP Echo Requests.

3.2 ICMP Sweep (Ping Sweep)

Querying multiple hosts using ICMP Echo requests is referred to as '*ICMP Sweep*' (or '*Ping Sweep*').

[19] Snort, written by Martin Roesch, can be found at http://www.snort.org.

[20] It is better to filter unwanted traffic at your border router, reducing traffic rates for your firewall.

For a small to midsize network the 'ping' utility is an acceptable solution to this kind of host detection, but with large networks (such as Class A, or a full Class B) this kind of scan is fairly slow mainly because 'ping' waits for a reply (or a time out to be reached) from the questionable IP address before proceeding to the next targeted IP address.

fping[21] is a UNIX utility which sends parallel mass ICMP Echo requests in a round robin fashion enabling it to be significantly faster than the usual 'ping' utility. It can also be fed with IP addresses with its accompanied tool gping. gping is used to generate a list of IP addresses which would be later fed into fping, directly or from a file, to perform the ICMP sweep. fping is also able to resolve hostnames of the probed machines if using the –d option.

Another UNIX tool that is able of doing an ICMP sweep in parallel, resolve the hostnames of the probed machines, save it to a file and a lot more is nmap[22], written by Fyodor.

For the Microsoft Windows operating system a notable ICMP sweep tool is Pinger from Rhino9[23], able of doing what fping and nmap do regarding this kind of scan.

Trying to resolve the names of the probed machines may discover the malicious computer attacker's IP address used for the probing, using the log of the authoritative DNS server of the targeted network.

The next example demonstrates the usage of nmap to perform an ICMP sweep against a calss C network. Our test lab contains several Microsoft Windows 2000 based machines, some Linux based machines, and couple of networking devices.

The –sP option instructs nmap to perform a 'ping scan'. The –PI option instructs nmap to send only true ICMP Echo requests. The default behavior when using the –sP option is to include the usage of TCP ACK host detection technique with 'regular' ICMP Echo requests.

```
[root@godfather /root]# nmap -sP -PI 172.18.2.1-254

Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host  (172.18.2.29) appears to be up.
Host x30.sys-security.com (172.18.2.30) appears to be up.
Host x31.sys-security.com (172.18.2.31) appears to be up.
Host x32.sys-security.com (172.18.2.32) appears to be up.
Host x34.sys-security.com (172.18.2.34) appears to be up.
Host x35.sys-security.com (172.18.2.35) appears to be up.
Host x36.sys-security.com (172.18.2.36) appears to be up.
Host  (172.18.2.38) appears to be up.
Host x40.sys-security.com (172.18.2.40) appears to be up.
Host x41.sys-security.com (172.18.2.41) appears to be up.
...

Nmap run completed -- 254 IP addresses (93 hosts up) scanned in 59
seconds
```

[15] ftp://ftp.tamu.edu/pub/Unix/src
[22] http://www.insecure.org
[23] The Rhino9 group no longer exists. Their tools are available from a number of sites on the Internet.

`nmap` will try to resolve host names by default. When it will fail we will see only the IP address in the output. If nmap succeed to resolve the IP address to a name than we will see both the name and the IP address of the target in the output.

If we wish to avoid the automatic resolving we should use the –n option.

```
[root@godfather /root]# nmap -n -sP -PI 172.18.2.1-254

Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host  (172.18.2.29) appears to be up.
Host  (172.18.2.30) appears to be up.
Host  (172.18.2.31) appears to be up.
Host  (172.18.2.32) appears to be up.
Host  (172.18.2.34) appears to be up.
Host  (172.18.2.35) appears to be up.
Host  (172.18.2.36) appears to be up.
Host  (172.18.2.38) appears to be up.
Host  (172.18.2.40) appears to be up.
Host  (172.18.2.41) appears to be up.
...

Nmap run completed -- 254 IP addresses (93 hosts up) scanned in 32
seconds
```

We can see that the results where produced faster without resolving.

ICMP sweeps should be easily detected by an intrusion detection systems (IDS) whether launched in the regular way, or if used in a parallel way.

Countermeasure: Block ICMP Echo requests coming from the Internet towards your network at your border router and/or Firewall. You can also configure your host(s) not to answer ICMP Echo Requests.

3.3 Broadcast ICMP

A simpler way to map a targeted network for alive hosts is by sending an ICMP Echo request to the broadcast address or to the network address of the targeted network.

The request would be broadcasted to all hosts on the targeted network. The alive hosts will send an ICMP Echo reply to the prober's source IP address (additional conditions apply here).

The malicious computer attacker has to send only one packet to produce this behavior.

This technique of host detection is applicable only to some of the UNIX and UNIX-like operating systems. Microsoft Windows based machines will not generate an answer (ICMP Echo reply) to an ICMP Echo request aimed at the broadcast address or at the network address of the network they reside on. They are configured not to answer those queries out-of-the box (This applies to all Microsoft Windows operating systems accept for Microsoft Windows NT 4.0 with service pack

below SP4). This is not an abnormal behavior as RFC 1122[24] states that if we send an ICMP Echo request to an IP Broadcast or IP Multicast addresses it *may* be silently discarded by a host.

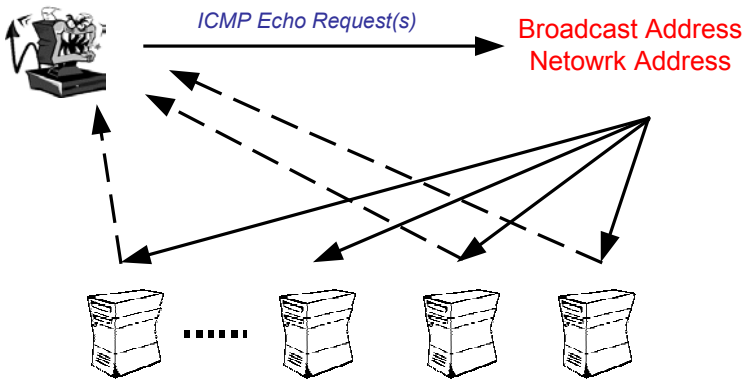


Figure 14: The Broadcast ICMP Technique

The next example demonstrates the behavior expected from hosts when sending an ICMP Echo request to the broadcast address of the network they reside on. The Linux based hosts on our test lab answered the query (172.18.2.200, 172.18.2.201), as well as the networking devices (172.18.2.29, 172.18.2.254). The Microsoft Windows 2000, and Microsoft Windows 2000 with SP1, silently ignored the request:

```
[root@localhost /root]# ping -b 172.18.2.255
WARNING: pinging broadcast address
PING 172.18.2.255 (172.18.2.255) from 172.18.2.201 : 56(84) bytes of
data.
64 bytes from 172.18.2.201: icmp_seq=0 ttl=255 time=6.380 msec
64 bytes from 172.18.2.200: icmp_seq=0 ttl=255 time=6.444 msec (DUP!)
64 bytes from 172.18.2.254: icmp_seq=0 ttl=255 time=6.469 msec (DUP!)
64 bytes from 172.18.2.29: icmp_seq=0 ttl=64 time=6.493 msec (DUP!)
...

--- 172.18.2.255 ping statistics ---
5 packets transmitted, 5 packets received, +15 duplicates, 0% packet
loss
round-trip min/avg/max/mdev = 5.629/5.875/6.493/0.299 ms
```

In the next example I have sent an ICMP Echo request to the network address of the targeted network. Here we can see that a slightly different behavioral pattern was produced. The Linux machines, and the Cisco Catalyst 6500 switch (172.18.2.254) answered our query while the other networking device did not produce an answer this time:

```
[root@godfather /root]# ping -b 172.18.2.0
WARNING: pinging broadcast address
PING 172.18.2.0 (172.18.2.0) from 172.18.2.201 : 56(84) bytes of data.
```

[24] RFC 1122: Requirements for Internet Hosts - Communication Layers, http://www.ietf.org/rfc/rfc1122.txt.

```
64 bytes from 172.18.2.201: icmp_seq=0 ttl=255 time=5.755 msec
64 bytes from 172.18.2.200: icmp_seq=0 ttl=255 time=6.034 msec (DUP!)
64 bytes from 172.18.2.254: icmp_seq=0 ttl=255 time=6.286 msec (DUP!)
...

--- 172.18.2.0 ping statistics ---
3 packets transmitted, 3 packets received, +6 duplicates, 0% packet
loss
round-trip min/avg/max/mdev = 4.395/5.185/6.286/0.648 ms
```

Note: Broadcast ICMP may result in a *Denial-Of-Service* condition if a lot of machines will respond to the query at once.

A more accurate table that lists which operating systems would answer to an ICMP Echo request aimed at their Network / Broadcast address is given below:

Operating System	Echo Request Broadcast
Linux Kernel 2.4.x	+
Linux Kernel 2.2.x	+
FreeBSD 4.0	-
FreeBSD 3.4	-
OpenBSD 2.7	-
OpenBSD 2.6	-
NetBSD	
Solaris 2.5.1	+
Solaris 2.6	+
Solaris 2.7	+
Solaris 2.8	+
HP-UX v10.20	+
Windows 95	-
Windows 98	-
Windows 98 SE	-
Windows ME	-
Windows NT 4 WRKS SP 3	-
Windows NT 4 WRKS SP 6a	-
Windows NT 4 Server SP4	-
Windows Family (including SP1)	-

Table 10: Which Operating Systems would answer to an ICMP ECHO Request aimed at the Broadcast Address of the Network they reside on?

Countermeasure: Block the IP directed broadcast on your border router. You can also configure your host(s) not to answer ICMP Echo Requests aimed at the Broadcast Address of the Network they reside on.

3.4 Non-ECHO ICMP

ICMP ECHO is not the only ICMP query message type available with the ICMP protocol.

Non-ECHO ICMP messages are being used for more advanced ICMP scanning techniques (not only probing hosts, but network devices, such as a router, as well).

The group of ICMP query message types includes the following:

ECHO Request (Type 8), and Reply (Type 0)
Time Stamp Request (Type 13), and Reply (Type 14)
Information Request (Type 15), and Reply (Type 16)
Address Mask Request (Type 17), and Reply (Type 18)
Router Solicitation (Type 10), and Router Advertisement (Type 9)

3.4.1 ICMP Time Stamp Request (Type 13) and Reply (Type 14)

The *ICMP Time Stamp Request and Reply* allows a node to query another for the current time. This allows a sender to determine the amount of latency that a particular network is experiencing. The sender initializes the identifier (used to identify Timestamp requests aimed at different destination hosts) and sequence number (if multiple Timestamp requests are sent to the same destination host), sets the originate time stamp and sends it to the recipient.

The receiving host fills in the receive and transmit time stamps, change the type of the message to time stamp reply and returns it to the recipient. The time stamp is the number of milliseconds elapsed since midnight UT (GMT).

The originate time stamp is the time the sender last touched the message before sending it, the receive time stamp is the time the recipient first touched it on receipt, and the Transmit time stamp is the time the receiver last touched the message on sending it.

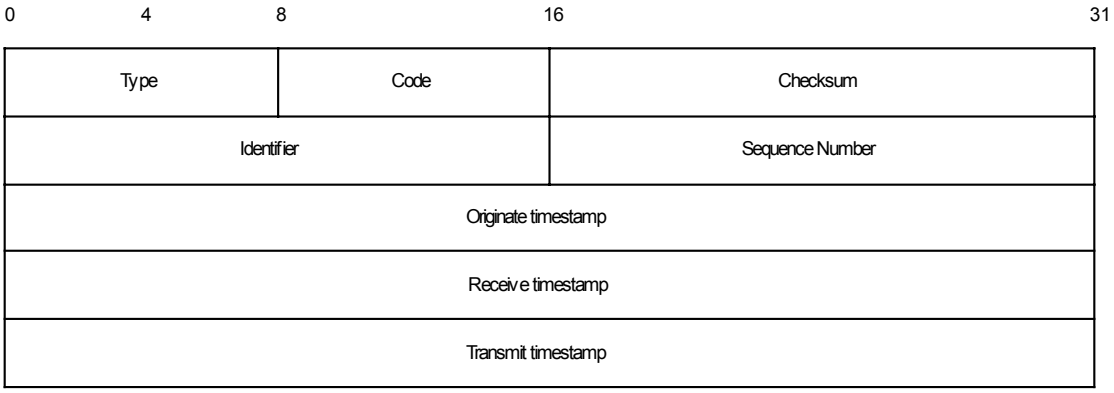


Figure 15: ICMP Time Stamp Request & Reply message format

As RFC 1122 state, a *host may* implement Timestamp and Timestamp Reply. If they are implemented a host must follow this rules:

- Minimum variability delay in handling the Timestamp request.
- The receiving host *must* answer to every Timestamp request that he receives.
- An ICMP Timestamp Request to an IP Broadcast or IP Multicast address *may* be silently discarded.
- The IP source address in an ICMP Timestamp reply *must* be the same as the specific-destination address of the corresponding Timestamp request message.

- If a source-route option is received in a Timestamp request, the return route *must* be reserved and used as a Source Route option for the Timestamp Reply option.
- If a Record Route and/or Timestamp option is received in a Timestamp request, this option(s) *should* be updated to include the current host and included in the IP header of the Timestamp Reply message.

Receiving an ICMP Timestamp Reply would reveal an alive host (or a networking device) that has implemented the ICMP Timestamp messages.

In the next example I have sent an ICMP Timestamp request, using the `sing`[25] utility, from a Linux host based on Kernel 2.4.2, to a host running Microsoft Windows 2000 professional. We are using the –c option to instruct `sing` how many requests it should send.

```
[root@godfather /root]# sing -c 1 -tstamp 172.18.2.149
SINGing to 172.18.2.149 (172.18.2.149): 20 data bytes
20 bytes from 172.18.2.149: seq=0 ttl=128 TOS=0 diff=2057048508

--- 172.18.2.149 sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```

The `snort` trace:

```
05/14/01-12:21:35.301542 172.18.2.201 -> 172.18.2.149
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:40
Type:13  Code:0  TIMESTAMP REQUEST
5A 05 00 00 02 02 26 46 00 00 00 00 00 00 00 00  Z.....&F........


05/14/01-12:21:35.301542 172.18.2.149 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:10964 IpLen:20 DgmLen:40
Type:14  Code:0  TIMESTAMP REPLY
5A 05 00 00 02 02 26 46 7C 9E 38 02 7C 9E 38 02  Z.....&F|.8.|.8.
```

Most of the operating systems have implemented the ICMP Timestamp request and reply mechanism. When I have sent an ICMP Timestamp request to a Windows NT 4 SP6a based machine, I got no reply. Again, this is not an abnormal behavior from the Microsoft Windows NT machine, just an implementation choice as RFC 1122 states.

Countermeasure: Block ICMP Time Stamp Requests coming from the Internet on the border Router and/or Firewall. If possible configure your host(s) to ignore ICMP Timestamp requests.

3.4.2 ICMP Information Request (Type 15) and Reply (Type 16)

The *ICMP Information Request/Reply* pair was intended to support self-configuring systems such as diskless workstations at boot time, to allow them to discover their network address.

The sender fills in the request with the Destination IP address in the IP Header set to zero (meaning this network). The request may be sent with both Source IP Address and Destination IP

[25] Sing, written by Alfredo Andreas Omella, can be found at www.sourceforge.net/projects/sing.

Address set to zero. The sender initializes the identifier and the sequence number, both used to match the replies with the requests, and sends out the request. The ICMP header code field is zero.

If the request was issued with a non-zero Source IP Address the reply would only contain the network address in the Source IP Address of the reply. If the request had both the Source IP Address and the Destination IP Address set to zero, the reply will contain the network address in both the source and destination fields of the IP header.

From the description above one can understand that the ICMP Information request and reply mechanism was intended to be used locally.

The RARP, BOOTP & DHCP protocols provide better mechanisms for hosts to discover its own IP address.

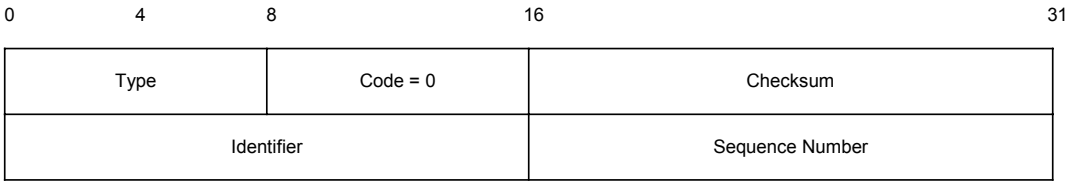


Figure 16: ICMP Information Request & Reply message format

The Information Request & Reply mechanism is now obsolete as stated in RFC 1122, and RFC 1812[26]. A *router should not* originate or respond to these messages; A *host should not* implement these messages.

Demands on one hand and reality on the other.

RFC 792 specifies that the Destination IP address should be set to zero, this mean that hosts that do not reside on the same network cannot send these ICMP query type.

But what would happen if we would send an ICMP Information Request with the Destination IP address set to a specific IP address of a host out in the void?

The next example illustrates that some operating systems would answer these queries even if not issued from the same network. The ICMP Information Request queries we are sending are not really RFC compliant because of the difference in the Destination IP address.

Those operating systems that answer our queries work in contrast to the RFC guidelines as well. We would see in the next example why.

In the next example I have sent an ICMP Information Request, using the `sing` utility, to an IBM AIX machine:

```
[root@aik icmp]# ./sing -info host_address[27]
SINGing to host_address (ip_address): 8 data bytes
```

[26] RFC 1812: Requirements for IP Version 4 Routers, http://www.ietf.org/rfc/rfc1812.txt . As the RFC states this mechanism is now obsolete - A *router should not* originate or respond to these messages; A *host should not* implement these messages.

[27] Since I have queried a production system for this test, with a permission of the owners, I do not wish to identify it.

```
8 bytes from ip_address: icmp_seq=0 ttl=238 Info Reply
8 bytes from ip_address: icmp_seq=1 ttl=238 Info Reply
8 bytes from ip_address: icmp_seq=2 ttl=238 Info Reply
8 bytes from ip_address: icmp_seq=3 ttl=238 Info Reply

--- host_address sing statistics ---
5 packets transmitted, 4 packets received, 20% packet loss
```

The `tcpdump` trace:

```
19:56:37.943679 ppp0 > x.x.x.x > y.y.y.y: icmp: information request
                       4500 001c 3372 0000 ff01 18a7 xxxx xxxx
                       yyyy yyyy 0f00 bee3 321c 0000
19:56:38.461427 ppp0 < y.y.y.y > x.x.x.x: icmp: information reply
                       4500 001c 661b 0000 ee01 f6fd yyyy yyyy
                       xxxx xxxx 1000 bde3 321c 0000
```

Lets do a quick analysis of the trace.

The ICMP Information request:

Value	Field	Additional Information
4	4-Bit Version	IP Version 4
5	4-Bit Header Length	4 x DWORD = 20 Bytes
00	8-Bit TOS	TOS=0
00 1c	16-Bit Total Length	
33 72	16-Bit Identification	
00 00	3-Bit Flags + 13-bit Fragment Offset	
ff	8-Bit TTL	TTL=255
01	8-Bit Protocol	1=ICMP
18 a7	16-Bit Header Checksum	
8b 5c d0 15	32-bit Source IP Address	139.92.208.21
xx xx xx xx	32-Bit Destination IP Address	
0f	8-Bit Type	Type=15
00	8-Bit Code	Code=0
be e3	16-Bit Checksum	
32 1c	16-Bit Identifier	
00 00	16-Bit Sequence Number	

The ICMP Information Reply:

Value	Field	Additional Information
4	4-Bit Version	IP Version 4
5	4-Bit Header Length	4 x DWORD = 20 Bytes
00	8-Bit TOS	TOS=0
00 1c	16-Bit Total Length	
66 1b	16-Bit Identification	
00 00	3-Bit Flags + 13-bit Fragment Offset	
ee	8-Bit TTL	TTL=238
01	8-Bit Protocol	1=ICMP
F6 fd	16-Bit Header Checksum	

Value	Field	Additional Information
xx xx xx xx	32-bit Source IP Address	
8b 5c d0 15	32-Bit Destination IP Address	139.92.208.21
10	8-Bit Type	Type=16
00	8-Bit Code	Code=0
bd e3	16-Bit Checksum	
32 1c	16-Bit Identifier	
00 00	16-Bit Sequence Number	

Instead of having the network address in the Source IP Address we are getting the IP address of the host.

Does the reply compliant with RFC 792 regarding this issue? Basically yes, because the RFC does not specify an accurate behavior.

The RFC states: “To form a information reply message, the source and destination addresses are simply reversed, the type code changes to 16, and the checksum recomputed”.

This means that if the ICMP Information Request is coming from outside (Destination is not zero) of the network in question, the network address would not be revealed. But still a host could be revealed if he answers the request.

The request is not compliant with the RFC in my opinion because it does not fulfill its job – getting the network address.

Countermeasure: Block ICMP Information Requests coming from the Internet on the border Router and/or Firewall.

3.4.3 ICMP Address Mask Request (Type 17) and Reply (Type 18)

The *ICMP Address Mask Request* (and Reply) is intended for diskless systems to obtain its subnet mask in use on the local network at bootstrap time. Address Mask request is also used when a node wants to know the address mask of an interface. The reply (if any) contains the mask of that interface.

Once a host has obtained an IP address, it could than send an Address Mask request message to the broadcast address of the network they reside on (255.255.255.255). Any host on the network that has been configured to send address mask replies will fill in the subnet mask, change the type of the message to address mask reply and return it to the sender[28].

RFC 1122 states that the Address Mask request & reply query messages are entirely optional.

[28] The usage of ICMP Address Mask request and reply mechanism was intended to be used on the local network the querying host resides on, only.

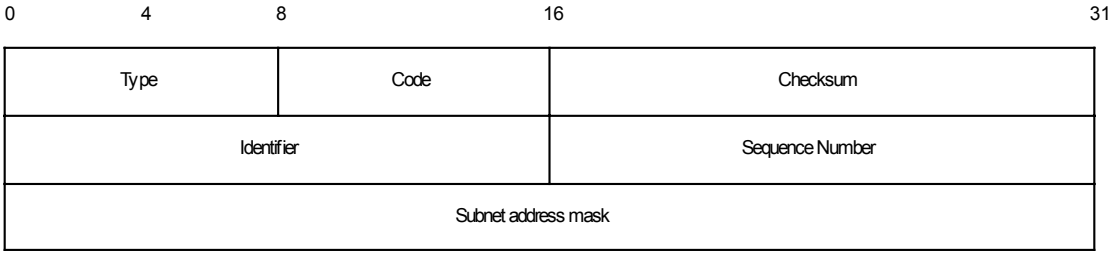


Figure 17: ICMP Address Mask Request & Reply message format

RFC 1122 also states that a system that has implemented ICMP Address Mask messages *must not* send an Address Mask Reply unless it is an authoritative agent for address masks.

Usually an Address Mask request would be answered by a gateway.

Receiving an Address Mask reply from a host would reveal an alive host that is an authoritative agent for address masks. It will also allow a malicious computer attacker to gain knowledge about your network's configuration. This information can assist the malicious computer attacker in determining your internal network structure, as well as the routing scheme.

Please note that a Router *must* implement ICMP Address Mask messages. This will help identify routers along the path to the targeted network (it can also reveal internal routers if this kind of traffic is allowed to reach them).

If a Router is following RFC 1812 closely, it should not forward on an Address Mask request to another network.

Not many operating systems answer to an ICMP Address Mask requests.

When I have tried to map which operating systems would answer (if at all) to an ICMP Address Mask requests, I have discovered that Sun Solaris is very cooperative with this kind of query:

```
[root@godfather /root]# sing -mask 172.18.1.15
SINGing to 172.18.1.15 (172.18.1.15): 12 data bytes
12 bytes from 172.18.1.15: seq=0 DF! ttl=254 TOS=0 mask=255.255.255.0
12 bytes from 172.18.1.15: seq=1 DF! ttl=254 TOS=0 mask=255.255.255.0
12 bytes from 172.18.1.15: seq=2 DF! ttl=254 TOS=0 mask=255.255.255.0
12 bytes from 172.18.1.15: seq=3 DF! ttl=254 TOS=0 mask=255.255.255.0

--- 172.18.1.15 sing statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
```

The `snort` trace:

```
05/14/01-12:24:19.211542 172.18.2.201 -> 172.18.1.15
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:32
Type:17  Code:0  ADDRESS REQUEST
5D 05 03 00 00 00 00 00                         ].......
```

```
05/14/01-12:24:19.211542 172.18.1.15 -> 172.18.2.201
ICMP TTL:254 TOS:0x0 ID:37780 IpLen:20 DgmLen:32 DF
Type:18  Code:0  ADDRESS REPLY
5D 05 03 00 FF FF FF 00                          ].......
```

We get another piece of information, not just the fact the host is reachable, but the address mask of the network the host resides on. Looking at the last example, we can conclude that the IP range of the network the host resides on is 172.18.1.1-255. Other reachable hosts might be out there…

Our last two examples are ICMP Address Mask requests aimed at a switch and at a router (which must implement ICMP Address Mask messages).

The following is an ICMP Address Mask request targeting a Cisco Catalyst 5505 with OSS v4.5:

```
inferno:/tmp# sing -mask -c 1 10.13.58.240
SINGing to 10.13.58.240 (10.13.58.240): 12 data bytes
12 bytes from 10.13.58.240: icmp_seq=0 ttl=60 mask=255.255.255.0

--- 10.13.58.240 sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
inferno:/tmp#


inferno:~# tcpdump -tnxv -s 1600 icmp
tcpdump: listening on xl0
10.13.58.199 > 10.13.58.240: icmp: address mask request (ttl 255, id
13170)
0000 :  4500 0020 3372 0000   FF01 FE99 0A0D 3AC7    E.. 3r........:.
0010 :  0A0D 3AF0 1100 6BF7   8308 0000 0000 0000    ..:...k.........

10.13.58.240 > 10.13.58.199: icmp: address mask is 0xffffff00 (ttl 60,
id 20187)
0000 :  4500 0020 4EDB 0000   3C01 A631 0A0D 3AF0    E.. N...<..1..:.
0010 :  0A0D 3AC7 1200 6BF6   8308 0000 FFFF FF00    ..:...k.........
0020 :  0000 0000 0000 0000   0000 0000 0000         ..............
^C
79 packets received by filter
0 packets dropped by kernel
inferno:~#
```

The last example is an ICMP Address Mask request sent to an Intel 8100 ISDN Router on another test network:

```
[root@aik icmp]# ./sing -mask 10.0.0.254
SINGing to 10.0.0.254 (10.0.0.254): 12 data bytes
12 bytes from 10.0.0.254: icmp_seq=0 ttl=64 mask=255.255.255.0
12 bytes from 10.0.0.254: icmp_seq=1 ttl=64 mask=255.255.255.0
12 bytes from 10.0.0.254: icmp_seq=2 ttl=64 mask=255.255.255.0

--- 10.0.0.254 sing statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
```

The `tcpdump` trace:

```
[root@aik /root]# tcpdump -x icmp
Kernel filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
16:34:30.666687  eth0  >  10.0.0.105  >  10.0.0.254:  icmp:  address  mask
request
                   4500 0020 3372 0000 ff01 7304 0a00 0069
                   0a00 00fe 1100 0afd e402 0000 0000 0000
16:34:30.667961  eth0  <  10.0.0.254  >  10.0.0.105:  icmp:  address  mask  is
0xffffff00
                   4500 0020 2cb7 0000 4001 38c0 0a00 00fe
                   0a00 0069 1200 0afc e402 0000 ffff ff00
                   0000 0000 0000 0000 0000 0000 0000
```

Countermeasure: Block ICMP Address Mask Requests coming from the Internet on the border Router and/or Firewall. If possible configure your host(s) to ignore ICMP Address Mask requests.

3.5 Non-ECHO ICMP Sweeps

We can query multiple hosts using a Non-ECHO ICMP query message type. This is referred as a Non-ECHO ICMP sweep.

Who would answer our query?

Hosts that answer to the following:

- Hosts that are in a listening state.
- Hosts running an operating system that implemented the Non-ECHO ICMP query message type that was sent.
- Hosts that are configured to reply to the Non-ECHO ICMP query message type (few conditions here as well, for example: RFC 1122 states that a system that implemented ICMP Address Mask messages *must not* send an Address Mask Reply unless it is an authoritative agent for address masks).

Given the conditions above, which host(s) would answer our queries?

Operating System	Info. Request	Time Stamp Request	Address Mask Request
Linux Kernel 2.4.x	-	+	-
Linux Kernel 2.2.x	-	+	-
FreeBSD 4.0	-	+	-
FreeBSD 3.4	-	+	-
OpenBSD	-	+	-
NetBSD			

Operating System	Info. Request	Time Stamp Request	Address Mask Request
Solaris 2.5.1	-	+	+
Solaris 2.6	-	+	+
Solaris 2.7	-	+	+
Solaris 2.8	-	+	+
HP-UX v10.20	+	+	-
AIX v4.x	+	+	-
ULTRIX 4.2 – 4.5	+	+	+
Windows 95	-	-	+
Windows 98	-	+	+
Windows 98 SE	-	+	+
Windows ME	-	+	-
Windows NT 4 WRKS SP 3	-	-	+
Windows NT 4 WRKS SP 6a		-	
Windows NT 4 Server SP 4	-	-	-
Windows 2000 Professional	-	+	-
Windows 2000 Server	-	+	-

Networking Devices	Info. Request	Time Stamp Request	Address Mask Request
Cisco Catalyst 5505 with OSS v4.5	+	+	+
Cisco Catalyst 2900XL with IOS 11.2	+	+	-
Cisco 3600 with IOS 11.2	+	+	-
Cisco 7200 with IOS 11.3	+	+	-
Intel Express 8100 ISDN Router	-	-	+

Table 11: non-ECHO ICMP Query of different Operating Systems and Networking Devices

Countermeasure: Block ICMP Information Requests, ICMP Address Mask Requests & ICMP Time Stamp Requests coming from the Internet on the border Router and/or Firewall.

3.6 Non-ECHO ICMP Broadcasts

We can send a Non-ECHO ICMP query message type to the broadcast address or to the network address of the targeted network.

The request would be broadcasted to all listening hosts on the targeted network.

Who would answer our query?

- Hosts that are in a listening state
- Hosts running an operating system that implemented the Non-ECHO ICMP query message type that was sent.
- Hosts that are configured to reply to the Non-ECHO ICMP query message type (few conditions here as well, for example: a host may discard Non-ECHO ICMP query message type requests targeted at the broadcast address. For example an ICMP Timestamp Request to an IP Broadcast or IP Multicast address *may* be silently discarded).

Given the conditions above, the answering hosts would almost always be UNIX and UNIX-like operating systems. Sun Solaris, HPUX, and Linux are the only operating systems, from the group of operating systems I have tested, that will answer to an ICMP Timestamp request aimed at the broadcast address of a network. HPUX would answer Information requests aimed at the broadcast address of a network. Non will answer to an ICMP Address Mask request aimed at the broadcast address of a network.

Operating System	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast
Linux Kernel 2.4.x	-	+	-
Linux Kernel 2.2.x	-	+	-
FreeBSD 4.0	-	-	-
FreeBSD 3.4			
OpenBSD 2.7	-	-	-
OpenBSD 2.6	-	-	-
NetBSD			
Solaris 2.5.1	-	+	-
Solaris 2.6	-	+	-
Solaris 2.7	-	+	-
Solaris 2.8	-	+	-
HP-UX v10.20	+	+	-
AIX 4.x			
ULTRIX 4.2 – 4.5			
Windows 95			
Windows 98	-	-	-
Windows 98 SE	-	-	-
Windows ME	-	-	-
Windows NT 4 WRKS SP 3	-	-	-
Windows NT 4 WRKS SP 6a			
Windows NT 4 Server SP 4	-	-	-
Windows 2000 Professional (& SP1)	-	-	-
Windows 2000 Server (& SP1)	-	-	-

Table 12: Operating Systems, which would answer to requests, aimed at the Broadcast address

Networking Devices	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast
Cisco Catalyst 5505 with OSS v4.5	+	+	+
Cisco Catalyst 2900XL with IOS 11.2	+	-	-
Cisco 3600 with IOS 11.2	+	-	-
Cisco 7200 with IOS 11.3	+	-	-
Intel Express 8100 ISDN Router	-	-	-

Table 13: Networking Devices, which would answer to requests, aimed at the Broadcast address

Countermeasure: Block the IP directed broadcast on the border router. Block ICMP Information Requests, ICMP Address Mask Requests & ICMP Time Stamp Requests coming from the Internet on the border Router and/or Firewall.

3.7 Host Detection Using ICMP Error Messages

For a malicious computer attacker any ICMP Error message received from a target network will have the same affect as of receiving an ICMP query reply message.

Sometimes the information with the ICMP error message, or the type of problem it represents will be more valuable information to the malicious computer attacker than with a usual ICMP query message reply.

For example receiving an ICMP Host Unreachable error message from a router will educate the malicious computer attacker that the IP address he tried to reach is either temporary down or not being used.

Another example might be with an ICMP Destination Unreachable port unreachable error message sent by the targeted IP address educating the malicious computer attacker that his attempt to reach a certain UDP port failed – the port is closed (and the targeted IP address is alive and reachable).

```
05/14/01-11:38:24.889109 172.18.1.2 -> 172.18.2.200
ICMP TTL:127 TOS:0x0 ID:58193 IpLen:20 DgmLen:56
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.200:1024 -> 172.18.1.2:53
UDP TTL:63 TOS:0x0 ID:19 IpLen:20 DgmLen:70
Len: 50
** END OF DUMP
00 00 00 00 45 00 00 46 00 13 00 00 3F 11 1F A6  ....E..F....?...
AC 12 02 C8 AC 12 01 02 04 00 00 35 00 32 9A 68  ...........5.2.h
```

Lets examine the next ICMP error message:

```
05/09/01-12:29:41.399543 RoutersIP -> SourceIP
ICMP TTL:244 TOS:0x0 ID:24442 IpLen:20 DgmLen:56
Type:3  Code:13  DESTINATION UNREACHABLE: PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
SourceIP:4667 -> DestinationIP:53
TCP TTL:53 TOS:0x0 ID:40019 IpLen:20 DgmLen:60
**U****F Seq: 0x97EABAF6  Ack: 0x1C1D1E1F  Win: 0x2223  TcpLen: 8
UrgPtr: 0x2627
** END OF DUMP
00 00 00 00 45 00 00 3C 9C 53 40 00 35 06 29 B0  ....E..<.S@.5.).
xx xx xx xx yy yy yy yy 12 3B 00 35 97 EA BA F6  .....Z...;.5....
```

This is an ICMP Destination Unreachable Communication Administratively Prohibited error message (type 3 code 13).

The ICMP error message advice the malicious computer attacker that a filtering device is present and filtering the destination system's network traffic. The filtering device is configured to block incoming TCP packets destined for port 53 on the targeted IP address.

It may help the malicious computer attacker to determine the type of the filtering device being used (whether this is a router/security device/another networking device), and to choose its tactics accordingly.

We can conclude that our destination host is up and running, but we cannot reach it, since the filtering device is blocking our packets, and instruct us to stop sending packets.

The ICMP error messages are not being intentionally triggered. They report non-transient error conditions for network traffic the malicious computer attacker has initiated.

In the next chapter I will discuss some advanced host detection methods based on attempts of the malicious computer attacker to trigger ICMP Error messages back from targeted IP addresses.

4.0 Advanced Host Detection using the ICMP Protocol

We will concentrate in the ability to trigger several types of ICMP error messages back from a targeted IP address (host).

We will force the target to generate an ICMP error message by mangling a certain field value in our query. We have several field values that we can choose from in order to generate several different ICMP error messages.

All conditions forced by the query host on the targeted IP address, will force the underlying OS kernel to issue an ICMP error message. With only one exception, all the error conditions will always trigger an ICMP error message.

This also lead us to use the advanced host detection methods in order to detect if a filtering device is present and forcing its filtering rules on the network traffic going to our targeted IP address (and probably on network traffic targeting the IP range of the network in question). The targeted host itself can force the filtering (host based firewall, for example), or it can be done by a networking device, or by another type of security device.

We can use the advanced host detection methods to detect access control lists (ACLs) forced by a filtering device on the protected network as well.

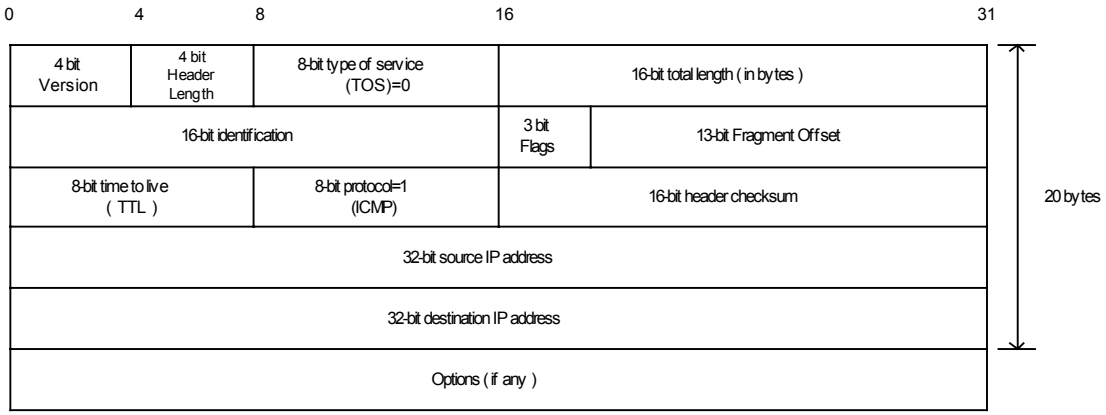


Figure 18: The IP Header

4.1 Triggering ICMP Parameter Problem error messages

An ICMP Parameter Problem error message is sent when a router (*must* generate this message) or a host (*should* generate this message) process a datagram and finds a problem with the IP header parameters, which is not specifically covered by another ICMP error message. The ICMP parameter problem error message is only sent if the error caused the datagram to be discarded.

To use this method we need to analyze the IP header and to decide what are the field values that can be mangled in our queries to trigger an ICMP parameter problem error message back from the targeted IP address (host).

We need to remember that from the list of fields that can be mangled, we need to choose only the fields, which do not have any other ICMP error message associated with.

This will force the targeted IP address to send back an ICMP parameter problem error message and to reveal its existence. We can receive two types of ICMP parameter problem error messages:

- Code 0 - The pointer field will point to the exact byte in the original IP Header, which caused the problem, or
- Code 2 - is sent when the header length or the total packet length values of the IP datagram do not appear to be accurate.

RFC 1812 requires a router to validate the following fields when processing a packet[29]:

- Checksum – a router must verify the IP checksum of any packet it received, and must discard messages containing invalid checksums.

According to RFC 1122 a host should check for validity of the following fields when processing a packet[30]:

- Version Number – if not 4 a host must silently discard the IP packet.
- Checksum – a host should verify the IP header checksum on every received datagram and silently discard every datagram that has a bad checksum.

It is possible to send an IP datagram with mangled IP header field values and still to get routed without getting dropped in the way to the probed machine. It should be noted that different routers perform different checks regarding the IP header field values (different implementation and interpretation of RFC 1812). When a router, because of a bad IP header field value, drops an IP packet and sends an ICMP parameter problem error message, it may be possible to identify the manufacture of the router, and to adjust the wrong IP header field value according to a field, which is not checked by the manufacture of that particular router.

A router may be more forgiving than a host regarding an IP header field value. This may result from the fact that a router is a vehicle for delivering the IP datagram and a host is the destination and the place where more processing on the datagram is being done.

The restrictions leave us with a number of fields only; some, which are crucial for our packet to arrive to its destination, will not be listed here:

- Header Length (already handled by code 2)
- TOS (Not relevant)
- Total Length (already handled by code 2)
- Identification (Not relevant)
- Flags (Not relevant)
- Fragment Offset
- Time to Live (errors reported by another ICMP error message)
- Protocol (errors reported by another ICMP error message)
- IP Options

The conditions outlined eliminate the usage of this method to a limited number of fields only. Practically to the header length, total datagram length, and to the IP option field values.

[29] RFC 1812 – Requirements for IPv4 Routers, http://www.ietf.org/rfc/rfc1812.txt.
[30] RFC 1122 – Requirements for Internet Host, http://www.ietf.org/rfc/rfc1122.txt.

Since we are locating the mangled field value in the IP header portion of the packet, we can carry any protocol with the triggering IP datagram.

This method is very powerful in detecting host(s) on the probed network with direct access from the Internet, since a host should generate this error message facing the conditions outlined. Routers must generate the ICMP parameter problem error message as well, this if they are the target of the probe.

The downside for this method is the detection. Intrusion Detection Systems *should* alert you about abnormalities in the attacked network traffic. It is not usual to see coming packets with bad IP headers field values, or to see ICMP parameter problem error messages leaving your network as response.

We can use this type of Host Detection method to sweep through the entire IP range of an organization and get back results, which will map all the hosts (and networking devices) on the probed network with direct access from the Internet.

Is a Filtering Device Present?

If a filtering device is protecting the targeted host we can detect its presence easily. Since we are using queries that require our targets to elicit an ICMP parameter problem error message back to us, than if we will not receive a reply back it will educate us that something suspicious is going on. Either the IP address is not being used, or a filtering device is filtering the traffic.

Even if a filtering device is protecting the targeted network (or the targeted IP), we can still try to send these forged packets. This time we will use more logic. We will use an underlying protocol and port that are likely to be allowed through by the filtering device ACL scheme. We can use for example TCP with ports 21,25,80; UDP port 53.

This will work because most of the firewalls in the market today will not validate if some field values are correct. One good example is the total IP datagram length field value. If the firewall can match its rule base with the query parameters, and its rule base allow the query, than the query will be allowed, and an error message will be produced[31].

An example is given here using the `isic` utility written by Mike Frantzen[32]. `isic` sends randomly generated packets to a target computer. Its primary uses are to stress test an IP stack, to find leaks in a firewall, and to test the implementation of Intrusion Detection Systems and firewalls. The user can specify how often the packets will be fragmented; have IP options, TCP options, an urgent pointer, etc.

In the next example I have sent 20 packets from a Linux based machine to a Microsoft Windows NT WRKS 4 SP4 based machine (the –p option with isic). The datagrams were not fragmented (the -F 0 option with isic) nor bad IP version numbers were sent (the -V 0 option with isic). The only weird thing sent inside the IP headers was random IP Header length values (the –l 100 option with `isic`), which have produced ICMP parameter problem Code 2 error message as I have anticipated.

[31] In my opinion Firewalls/Filtering Devices should check the validity of those fields used to elicit the ICMP Parameter Problem error message and disallow this kind of traffic.
[32] http://expert.cc.purdue.edu/~frantzen/

```
[root@stan packetshaping]# ./isic -s 192.168.5.5 -d 192.168.5.15 -p 20
-F 0 -V 0 -I 100
Compiled against Libnet 1.0
Installing Signal Handlers.
Seeding with 2015
No Maximum traffic limiter
Bad IP Version   = 0%                Odd IP Header Length     = 100%
Frag'd Pcnt     = 0%

Wrote 20 packets in 0.03s @ 637.94 pkts/s
```

The `tcpdump` trace:

```
12:11:05.843480   eth0   >   kenny.sys-security.com   >   cartman.sys-
security.com: ip-proto-110 226 [tos 0xe6,ECT]   (ttl 110, id 119,
optlen=24[|ip])

12:11:05.843961   eth0   P   cartman.sys-security.com   >   kenny.sys-
security.com: icmp: parameter problem - octet 21 Offending pkt:
kenny.sys-security.com > cartman.sys-security.com: ip-proto-110 226
[tos 0xe6,ECT] (ttl 110, id 119, optlen=24[|ip]) (ttl 128, id 37776)
```

An incorrect usage of the IP option field values will almost always trigger an ICMP Parameter Problem error message.

4.1.1 ACL Detection

We can use this host detection method to detect an ACL scheme enforced be a filtering device on a protected network.

With this type of query any protocol can be embedded inside the offending packet. We can use all available combinations of protocols and type of messages (ports for UDP and TCP, type and code with ICMP), on the entire IP range of a targeted network.

We need to mangle the offending packet wisely.

If we will send a bad IP header length value, than most of the firewall in the market today, will drop the query when they examine it. They will not be able to match their rule base with the query. This is because some of the parameters the firewall will look for could not be matched, or they reside beyond the IP header borders. I can name the destination port and source port with UDP and TCP, or the type and code fields with ICMP for example (if a longer false value is given).

So IP header length is out of the question. We are left we two IP header field values:

- Total Length
- IP Options

Some firewalls in the market today, will drop any packet that has an IP option value carried with it. The reason is that some firewalls will not intelligently parse the IP options.

We are left only with the total length field value. The mangled value we should send in this field should trigger the host to send back an ICMP parameter problem error message. It is also required that the firewall will be able to access the information it needs to match the packet against its rule base. This means that a mangled total length field value can be operating only on the data portion (and beyond) of the underlying protocol used.

If we will claim that the packet is smaller than it really is, than in nearly all cases nothing will happen. For example we can take an ICMP Echo request query with no data carried with it. It is still regarded as legitimate traffic (this is the way some tools act, like nmap and hping2).

We can only send a total IP datagram field value that will claim that our packet is bigger than it really is. The host will try to grab the data from the area, which is not there, and will issue an ICMP Parameter Problem Code 2 error message back to the querying IP address.

It will pass the firewall (if the ACL allows it), hit the host, and generate the error message back to the querying IP address.

If we probe the entire IP range of a targeted network with all possible combinations of protocols and services (ports/types and codes), it would draw us the targeted network topology map, and will allow us to determine the access list (ACL) a filtering device (if present, and not blocking outgoing ICMP Parameter Problem error messages) is forcing on the targeted network.

4.1.1.1 ACL Detection - An example with ICMP as the underlying Protocol

When the embedded protocol inside the offending packet is ICMP, we will query the targeted network with all possible combinations of IP addresses and ICMP query message types.

If we will receive a reply from a certain IP address in the targeted network IP range, it will educate us that we have a host that is reachable from the Internet, with a certain type of ICMP query message that was embedded inside the offending packet (we get this information back in the ICMP error message).

It will indicate that the ICMP query message type is allowed through the access control list (ACL) rules to that certain IP address, and that ICMP parameter problem error messages are allowed to be sent from the queried IP address to the Internet.

We might have several reasons not to receive an ICMP parameter problem error message back from the targeted IP address:

- The Filtering Device validates the 'total length' field value against the actual number of bytes it receives for that packet.
- The Filtering Device is filtering the type of the ICMP message we are using.
- The Filtering Device blocks ICMP Parameter Problem error messages initiated from the protected network destined to the Internet.

4.1.1.2 ACL Detection – An example with TCP/UDP as the underlying protocol

When the embedded protocol inside the offending packet is either UDP or TCP, we will query the targeted network with all possible combinations of IP addresses and TCP/UDP ports.

If we will receive a reply from a certain IP address in the targeted network IP range, it will educate us that we have a host that is reachable from the Internet, with the TCP/UDP protocol using port *z* (the port that was used for that probe) that was embedded inside the offending packet (we get this information back in the ICMP error message).

It will indicate that the TCP/UDP protocol using port z is allowed through the access control list (ACL) rules to that certain IP address, and that ICMP Parameter Problem error messages are allowed to be sent from the queried IP address to the Internet.

We might have several reasons not to receive an ICMP parameter problem error message back from the targeted IP address:

- The Filtering Device validates the 'total length' field value against the actual number of bytes it receives for that packet.
- The Filtering Device filters the Protocol used.
- The Filtering Device is filtering the specific port we are using for the probe.
- The Filtering Device blocks ICMP Parameter Problem error messages initiated from the protected network destined to the Internet. In our case, the filtering device may be blocking the specific host we are probing for outgoing ICMP Parameter Problem datagrams.

Countermeasure: Block outgoing ICMP Parameter Problem error messages coming from a protected network targeting hosts on the Internet on the Firewall & on the border Router.

Check with the manufacture of your filtering device which fields it really validates on the IP header when processing a datagram.

4.2 IP Datagrams with not used field values

The next host detection method is based on our ability to mangle some IP header field values, and introduce values, which will trigger ICMP Destination Unreachable error messages of certain types back from a probed IP address. This is simply because the values which we will be using are not being used on the targeted host.

What are the fields we can use for this method?
A *destination host* issues a destination unreachable message when the protocol specified in the *protocol number* field of the original datagram is not active on the destination host, or the *specified port* is inactive.

4.2.1 The Protocol Field example[33]

4.2.1.1 Using non-Used IP protocol values

If we will use a value, which does not represent a valid protocol field number being used on a targeted machine, the targeted machine will elicit an ICMP Destination Unreachable Protocol Unreachable error message back to us.

[33] Note that some hosts (AIX, HP-UX, Digital UNIX) may not send ICMP Protocol Unreachable error messages.

By sending crafted packets of this kind to all IP addresses within the IP address range of a targeted network we can map the hosts and networking devices that are reachable from the Internet (assuming no filtering device is present, or filtering the specific traffic).

IANA, the Internet Assign Number Authority, maintain the protocol values. The full list is available from: http://www.isi.edu/in-notes/iana/assignments/protocol-numbers.

4.2.1.1.1 Detecting if a Filtering Device is present

A packet sent with a protocol value, which does not represent a valid protocol field number being used on the targeted machine, should elicit an ICMP Destination Unreachable Protocol Unreachable error message from a targeted machine. Since the value we are using does not represent a valid protocol being used on the targeted machine it will elicit an ICMP protocol unreachable error message from each and every machine probed with this kind of scan. This is true unless the targeted IP address underlying operating system is AIX, HP-UX, or Digital UNIX. If a reply is not received we can assume that a filtering device prevents our packet from reaching our destination or from the reply to reach the Internet.

4.2.1.2 `Protocol Scan`

We can use this method in order to examine which protocols are being used on a targeted machine.

We will use all of the combinations available for the IP protocol field value, and since the IP protocol field has only 8 bits in length, there could be 256 combinations available.

If we will not receive an ICMP protocol unreachable error message back from the targeted host, for the field value we were using in our query it will educate us that this field value represents a valid protocol, which is being used on the targeted machine.

If we will receive an ICMP protocol unreachable error message back from the targeted host, for the field value we were using in our query it will educate us that this field value is not being used on the targeted machine.

From the answers/no-answers we have received we could than combine a list of available protocols on the targeted machine.

`nmap` 2.54 beta 1 has integrated this method of scanning and Fyodor has named it “IP Protocol scan”. `nmap` sends raw IP packets *without any further protocol header* (no payload) to each specified protocol on the target machine. If an ICMP Protocol Unreachable error message is received, the protocol is not in use. Otherwise it is assumed it is opened (or a filtering device is dropping our packets).

If our goal was Host Detection only, than using the `nmap` implementation would be a bit of an overkill.

If we wish to use this scan type for other purposes, such as ACL scheme detection, than we would need the payload data as well.

Not having any payload with our query using `nmap` will turn this type of scan quite easily.

A firewall might block the queries initiated with nmap since there is no protocol header (even for TCP/UDP/ICMP/IGMP) carried with the queries and the firewall cannot match the query with the

firewall's rule base. In this circumstance we will have all 256 possible protocol values seems as being used on the targeted machine.

In the next example I have used `nmap` 2.54 beta 22 in order to scan a Microsoft Windows 2000 SP1 Professional based machine:

```
[root@godfather /root]# nmap -vv -sO 172.18.2.200
Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host hostname (172.18.2.200) appears to be up ... good.
Initiating IPProto Scan against hostname (172.18.2.200)
The IPProto Scan took 4 seconds to scan 254 ports.
Interesting protocols on hostname (172.18.2.200):
(The 249 protocols scanned but not shown below are in state: closed)
Protocol   State       Name
1          open        icmp
2          open        igmp
6          open        tcp
17         open        udp
47         open        gre


Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

A `snort` trace of some of the communication exchanged:

```
05/20/01-13:09:24.502761 172.18.2.201 -> 172.18.2.200
PROTO176 TTL:47 TOS:0x0 ID:8652 IpLen:20 DgmLen:20


05/20/01-13:09:24.502761 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:15672 IpLen:20 DgmLen:56
Type:3  Code:2  DESTINATION UNREACHABLE: PROTOCOL UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.201 -> 172.18.2.200
PROTO176 TTL:47 TOS:0x0 ID:8652 IpLen:20 DgmLen:20
Protocol: 0xB0 (unknown or header truncated)** END OF DUMP
00 00 00 00 45 00 00 14 21 CC 00 00 2F B0 0B B9  ....E...!.../...
AC 12 02 C9 AC 12 02 C8 02 C8 02 C8 02 C8 02 C8  ................


05/20/01-13:09:24.502761 172.18.2.201 -> 172.18.2.200
IPCOMP TTL:47 TOS:0x0 ID:7050 IpLen:20 DgmLen:20


05/20/01-13:09:24.502761 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:15678 IpLen:20 DgmLen:56
Type:3  Code:2  DESTINATION UNREACHABLE: PROTOCOL UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.201 -> 172.18.2.200
IPCOMP TTL:47 TOS:0x0 ID:7050 IpLen:20 DgmLen:20
Protocol: 0x6C (unknown or header truncated)** END OF DUMP
00 00 00 00 45 00 00 14 1B 8A 00 00 2F 6C 12 3F  ....E...../l.?
AC 12 02 C9 AC 12 02 C8 02 C8 02 C8 02 C8 02 C8  ................
```

Is a Filtering Device Present?

With the 'protocol scan' identifying the presence of a firewall is easy. We need to choose a protocol number, which is not being used. We can look at the IANA list (http://www.isi.edu/in-notes/iana/assignments/protocol-numbers) and pick a number from there.

Our query, using the unused protocol number, should elicit an ICMP Protocol Unreachable error message back from the targeted IP address, unless the targeted IP address is AIX, HPUX, or Digital Unix. If no ICMP Protocol Unreachable error message is received than a firewall is present and filtering the traffic going to the targeted IP address, or our target IP address is either AIX, HPUX or Digital Unix.

In the next example I have tried to scan a Sun Solaris 2.7 based machine sitting behind a Check Point FW-1 v4.1 SP3, using `nmap` 2.54 beta 22:

```
[root@godfather /root]# nmap -vv -sO IP_Address
Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host hostname (IP_Address) appears to be up ... good.
Initiating IPProto Scan against hostname (IP_Address)
The IPProto Scan took 16 seconds to scan 254 ports.
Interesting protocols on hostname (IP_Address):
Protocol   State       Name
1          open        icmp
2          open        igmp
3          open        ggp
4          open        ip
5          open        st
6          open        tcp
7          open        cbt
8          open        egp
9          open        igp
10         open        bbn-rcc-mon
...
17         open        udp
...
36         open        xtp
37         open        ddp
38         open        idpr-cmtp
39         open        tp++
40         open        il
41         open        ipv6
55         open        mobile
...
141        open        unknown
142        open        unknown
...
252        open        unknown
253        open        unknown
254        open        unknown


Nmap run completed -- 1 IP address (1 host up) scanned in 16 seconds
```

Since nmap produce the packets for this type of scan without any payload, we would expect any firewall product, which is configured correctly, to drop any packet. This is since the firewall will not be able to match all the parameters it needs to verify the traffic against its rule base.

ACL Detection with the 'Protocol Scan' – Not Really

If we wish to use the 'protocol scan' for ACL scheme detection, I am in doubt it will be the best method to use.

All operating systems use the icmp, udp, and tcp protocols. If we wish to query for another protocol availability on a targeted IP address, we can use the 'protocol scan' and than use another method to check what is valid with this type of protocol.

Another aspect is that we have at least three operating systems which do not produce ICMP protocol unreachable error messages.

Countermeasure: Block outgoing ICMP Protocol Unreachable error messages coming from the protected network to the Internet on your Firewall and/or Border Router.

If you are using a firewall check that your firewall block protocols, which are not supported according to IANA (deny all stance).

4.3 Abusing IP fragmentation

When a host receives a fragmented datagram with some of its pieces missing, and does not get the missing part(s) within a certain amount of time the host will discard the datagram and generate an ICMP Fragment Reassembly Time Exceeded error message back to the sending host.

We can use this behavior as a Host Detection method, by sending fragmented datagrams with missing fragment(s) to a targeted host, and wait for an ICMP Fragment Reassembly Time Exceeded error message to be received from a targeted host(s), if any.

When we are using this method against the IP range of a targeted network, we will be able to discover the network topology of that targeted network.

In the next example I have sent a TCP fragment from my Linux based machine to a Microsoft Windows ME based machine. I was using the `hping2`[34] utility to generate the query (-x option to generate a fragment):

```
[root@godfather bin]# hping2 -c 1 -x -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): NO FLAGS are set, 40 headers + 0 data
bytes

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

[34] HPING2 written by antirez, http://www.kyuzz.org/antirez/hping/ .

The `tcpdump` trace:

```
20:20:00.226064     ppp0     >     x.x.x.x.1749     >     y.y.y.y.0:     .
1133572879:1133572879(0) win 512 (frag 31927:20@0+) (DF) (ttl 64)
                    4500 0028 7cb7 6000 4006 c8fd xxxx xxxx
                    yyyy yyyy 06d5 0000 4390 f30f 0c13 6799
                    5000 0200 27a8 0000

20:21:00.033209 ppp0 < y.y.y.y > x.x.x.x: icmp: ip reassembly time
exceeded Offending pkt: [|tcp] (frag 31927:20@0+) (DF) (ttl 55) (ttl
119, id 12)
                    4500 0038 000c 0000 7701 6e9e yyyy yyyy
                    xxxx xxxx 0b01 b789 0000 0000 4500 0028
                    7cb7 6000 3706 d1fd xxxx xxxx yyyy yyyy
                    06d5 0000 4390 f30f
```

Is a Filtering Device Present?

It is possible to detect if a firewall is present and forcing its rule base on a targeted network using the IP fragmentation abuse.

The behavioral pattern, when not receiving some fragments of the original datagram in a certain time frame, will always be the same on each and every operating system. This means that all will issue an ICMP fragment reassembly time exceeded error message back to the querying host.

If we will send one or few fragments of a datagram only to a targeted IP address, and not receive any reply back it will educate us that there is a filtering device present which prevents our query to reach the targeted IP address, or prevents the ICMP error message from reaching the Internet.

There is always the possibility where the targeted IP address is not available as well.

4.3.1 ACL Detection

The method of abusing fragmentation can be used not only to map the entire topology map of a targeted network, but also to determine an ACL scheme a firewall or another filtering device is forcing on a protected network.

We will have to query the entire IP range of a targeted network with all combinations possible for transport protocols (UDP and TCP) and ports, and for ICMP and codes. The query will be sent fragmented, where only some of the fragments will be sent, but not all.

With this method we need to slice our offending packet(s) wisely, since firewalls (and other filtering devices) might block fragmentation occurring in the first packet of communication if the fragmentation occurs 'too early'. For example, if we will fragment the first TCP packet starting the TCP handshake, and will not include the TCP flags section inside the fragmented packet, than most of the firewalls in the market today will drop the connection attempt. Some of them will do so instantly, while other firewalls will store the fragment we have just sent until we will send the missing pieces or a time limit will be reached. This might happen with any fragmentation of the initiating TCP handshake.

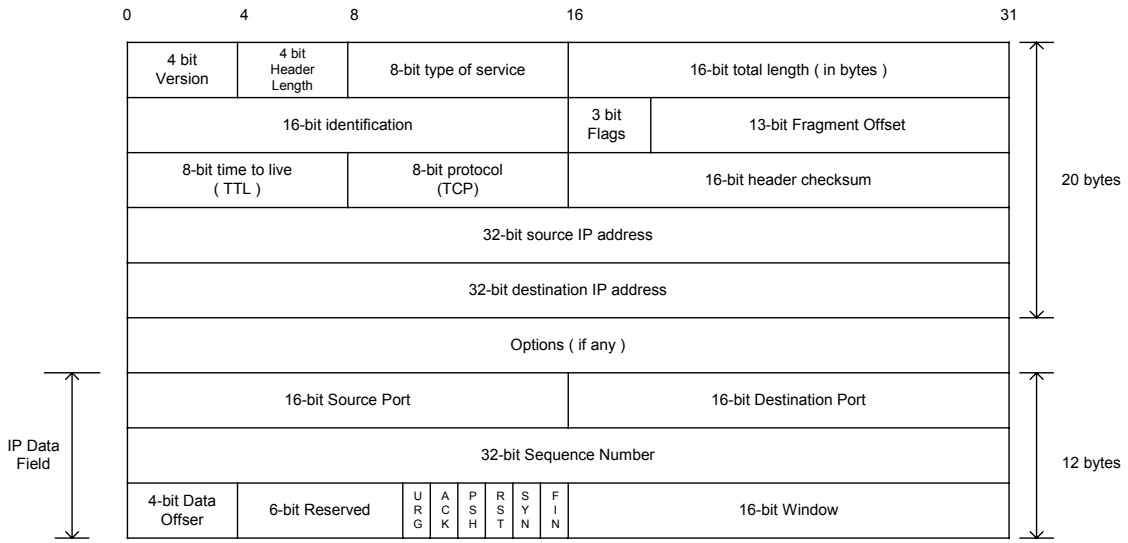


Figure 19: An Example: A TCP packet fragmented after only 12 bytes of TCP information

We might have better luck if we will be using the UDP transport protocol, since it is a stateless protocol. If we will 'slice' the UDP datagram after the relevant information to be matched by the firewall to its rule base, than we might succeed.

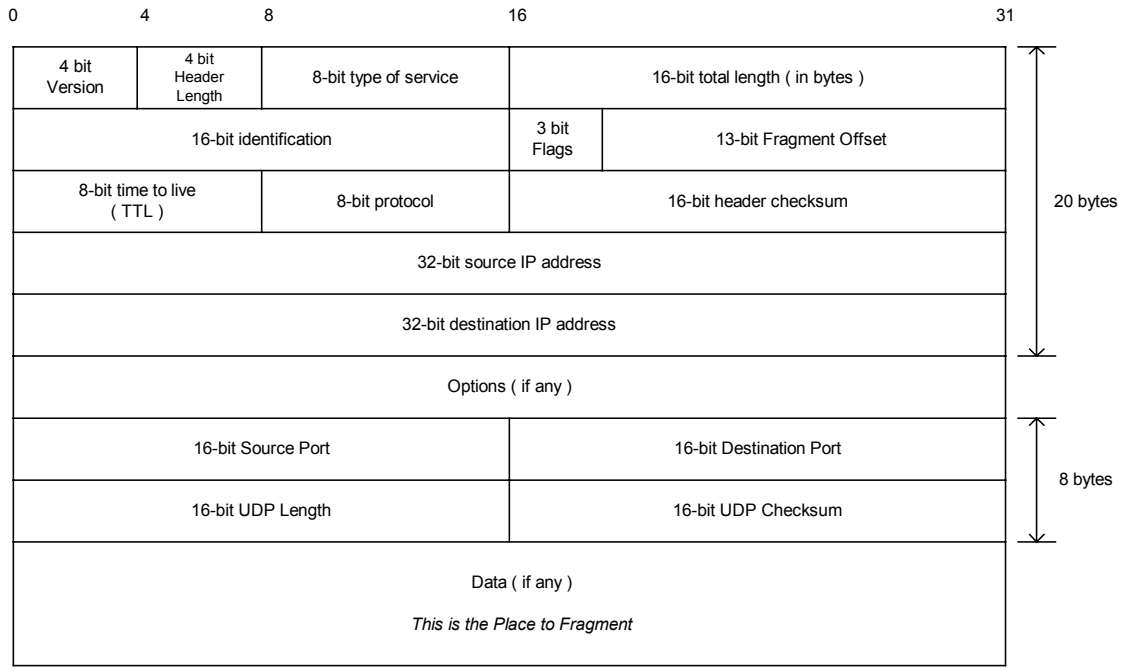


Figure 20: An Example with UDP. Slicing should occur in the Data portion

We can use the same 'slicing' method with ICMP and slice the query in the ICMP data portion. But please bare in mind that there are ISPs, which do not route ICMP fragmented datagrams.

When we will receive a reply from one of the targeted IP addresses it will educate us that we have a host, which is reachable via the Internet with the protocol and port used, and an ACL scheme which allows this type of communication (as well as the ICMP Fragment Reassembly Time Exceeded error message to be sent from the protected network to the Internet).

If we will not get any reply from a targeted IP address we have queried we might conclude that:

- The filtering device is filtering the Protocol used.
- The filtering device is filtering the specific port we are targeting.
- The filtering device blocks ICMP Fragment Reassembly Time Exceeded error messages initiated from the protected network destined to the Internet.

Countermeasure: Block outgoing ICMP Fragment Reassembly Time Exceeded Error messages from your protected network to the Internet.

4.4 Using UDP Scans (or why we wait for the ICMP Port Unreachable)

With this method we are abusing UDP to perform a scan. When we try to communicate with a closed UDP port we will receive an ICMP Port Unreachable error message back from the targeted host. If the port we were trying to connect to is in listening state than no reply will be generated, since UDP is a stateless protocol.

Is a Filtering Device Present?

When a filtering device is blocking UDP traffic aimed at a targeted IP address it will copycat the behavior pattern as with an open UDP port. We will not receive any reply back.

If we will query a large number of UDP ports on the same host and will not receive a reply from a large number of ports, it will look like a large number of queried UDP ports are opened, while a filtering device is probably blocking the traffic and nearly all of the ports are closed.

How can we remedy this?
We can set a threshold number of non-answering UDP ports, when reached we will assume a filtering device is blocking our probes.

Fyodor has implemented a threshold with nmap 2.3 beta 13, so when performing a UDP scan and not receiving an ICMP protocol unreachable error message back from a certain number of ports, it would assume a filtering device is monitoring the traffic, rather than reporting those ports as opened.

4.4.1 A Better Host Detection Using UDP Scan

We will take the UDP scan method and tweak it a bit for our needs. We know that a closed UDP port will generate an ICMP Port Unreachable error message indicating the state of the port - closed UDP port. We will choose a UDP port that should be definitely closed (according to the IANA list of assigned ports ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers). For example we can use port 0 (but it would reveal our probe pretty easily).

Based on the fact that sending a UDP datagram to a closed port should elicit an ICMP Port Unreachable, we would send one datagram to the port we have chosen, than:

- If no filtering device is present we will receive an ICMP Port Unreachable error message, which will indicate that our targeted Host is alive (or if this traffic is allowed by the filtering device).
- If no answer is received – a filtering device is filtering that port.

Instead of using port 0 we can choose a number of closed UDP ports according to IANA's port list. In each query we will be using another port so detection will be harder.

In the next example, using the `hping2` utility, I have tried to connect to a closed UDP port (port 50) on the host 172.18.2.131:

```
[root@pooh /root]# hping -2 -c 2 -p 50 172.18.2.131
eth0 default routing interface selected (according to /proc)
HPING 172.18.2.131 (eth0 172.18.2.131): udp mode set, 28 headers + 0
data bytes
ICMP Port Unreachable from 172.18.2.131  (unknown host name)
ICMP Port Unreachable from 172.18.2.131  (unknown host name)

--- 172.18.2.131 hping statistic ---
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The `snort` trace:

```
05/20/01-12:48:37.553394 172.18.2.200:1778 -> 172.18.2.131:50
UDP TTL:64 TOS:0x0 ID:34904 IpLen:20 DgmLen:28
Len: 8


05/20/01-12:48:37.553580 172.18.2.131 -> 172.18.2.200
ICMP TTL:128 TOS:0x0 ID:11214 IpLen:20 DgmLen:56
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.200:1778 -> 172.18.2.131:50
UDP TTL:64 TOS:0x0 ID:34904 IpLen:20 DgmLen:28
Len: 8
** END OF DUMP
00 00 00 00 45 00 00 1C 88 58 00 00 40 11 95 09  ....E....X..@...
AC 12 02 C8 AC 12 02 83 06 F2 00 32 00 08 9B 4A  ...........2...J
```

We can use the not used UDP port number we have chosen, or a list of UDP ports that are likely not being used, and query all the IP range of an attacked network. Getting a reply back would reveal a live host. No reply would mean a filtering device is covering those hosts UDP traffic, and probably other protocols and hosts as well.

4.5 Using Packets bigger than the PMTU of internal routers to elicit an ICMP Fragmentation Needed and Don't Fragment Bit was Set (configuration problem)

If internal routers have a Path-MTU that is smaller than the Path-MTU for a path going through the border router, those routers would elicit an ICMP "Fragmentation Needed and Don't Fragment Bit was Set" error message back to an initiating host if receiving a packet too big to process (but small enough to path through the border router) that has the Don't Fragment Bit set with the IP Header, discovering internal architecture of the router deployment of the attacked network.

This is, in my opinion, a configuration problem causing a security hazard.

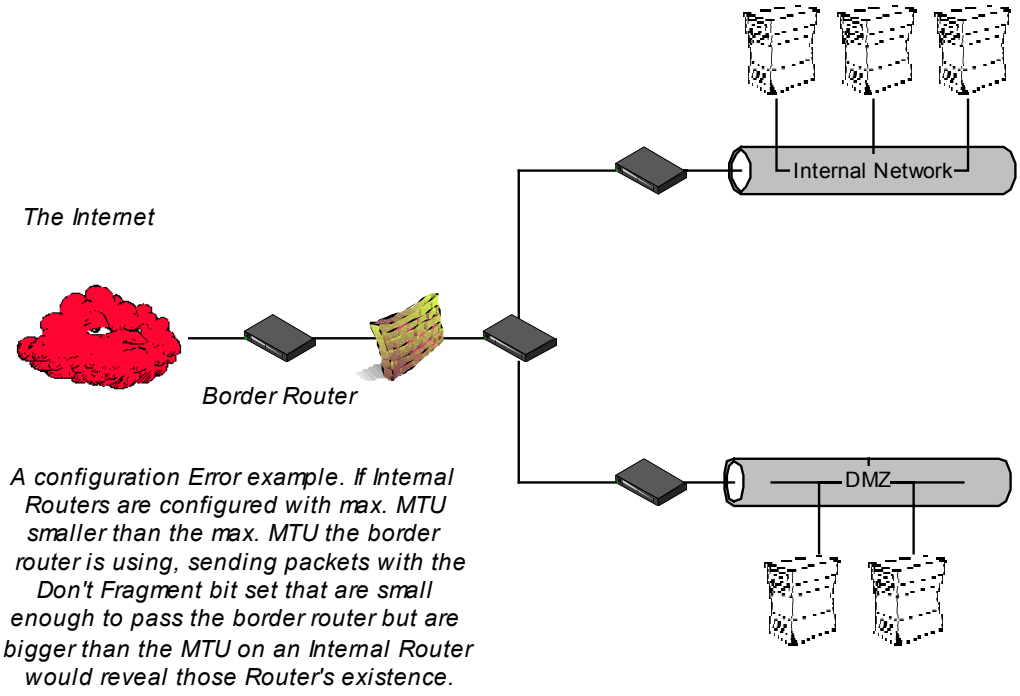


Figure 21: Using Packets bigger than the PMTU of internal routers to elicit an ICMP Fragmentation Needed and Don't Fragment Bit was Set

5.0 Inverse Mapping

Inverse Mapping is a technique used to map internal networks or hosts that are protected by a filtering device[35]. Usually some of those systems are not reachable from the Internet. We use routers, which will give away internal architecture information of a network, even if the question they were asked does not make any sense, for this scanning type. We compile a list of IP's that list what is not there, and use it to conclude were things probably are.

We send a number of packets to different IP's we suspect are in the IP range of a network we are targeting. When a router, either an exterior or interior, gets these packets for further processing, it looks at the IP address and makes decisions of routing based on it solely. When a router gets a packet with an IP address which is not used in the IP space / network segment of the part of the targeted network he serves, the router will elicit an ICMP Host Unreachable (generated by a router if a route to the destination host on a directly connected network is not available – the destination host does not respond to ARP request) or ICMP Time Exceeded error message(s) (because processing time took too long, and in the mean time the TTL has reached zero) back to the offending packet's source IP address. If we do not get an answer about a certain IP address (or the targeted IP address answered our query) we can assume this IP exist inside the probed network[36].

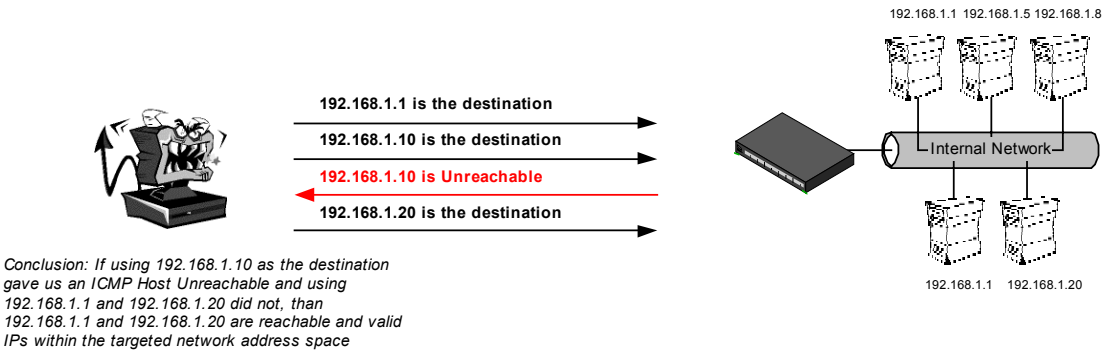


Figure 22: The Inverse Mapping Logic

5.1 Inverse Mapping Using ICMP Query Request(s), and ICMP Query Reply(s)

Theoretically speaking, using any ICMP query message type or any ICMP query reply message type in order to inverse map a network using a router is possible.

With the next example I have sent an ICMP Echo request to an IP address, which is part of the IP address range of a 'targeted network':

```
[root@cartman]# ./icmpush -vv -echo Target_IP[37]
 -> Outgoing interface = 192.168.1.5
```

[35] Usually it will be a Router with an Access Control List.

[36] There is also a possibility that a filtering device is blocking our probes, or the replies.

[37] The real IP's of the targeted host and the Router were replaced because of legal problems that might arise when the ISP's personal that was used would understand it was one of their Routers used for this experiment.

```
 -> ICMP total size = 12 bytes
 -> Outgoing interface = 192.168.1.5
 -> MTU = 1500 bytes
 -> Total packet size (ICMP + IP) = 32 bytes
ICMP Echo Request packet sent to Target_IP (Target_IP)

Receiving ICMP replies ...
-------------------------------------------------------
Routers_IP  ...
         Type = Time Exceeded (0xB)
   Code = 0x0     Checksum = 0xF98F
     Id = 0x0         Seq# = 0x0
-------------------------------------------------------
./icmpush: Program finished OK


ICMP TTL:254 TOS:0x0 ID:13170
ID:12291   Seq:317  ECHO


02/13-09:16:31.724400 Routers_IP -> 192.168.1.5
ICMP TTL:57 TOS:0x0 ID:7410
TTL EXCEEDED
```

The last hop router has issued an ICMP time to leave exceeded in transit error message. The router has failed to deliver the query to its destination since the processing time limit has been reached while waiting for an answer to its arp request looking for the physical address of the interface that represents the targeted IP address.

If a filtering device is protecting a targeted network, and configured correctly, than ICMP Echo replies will be blocked and dropped. Since many firewalls do not have the ability of dynamic filter / statefull inspection with ICMP, and the functionality of the 'ping' utility initiated from a protected network destined the Internet is required for troubleshooting purposes, for example, than ICMP Echo reply will be allowed to enter the protected network from the Internet. This will enable ICMP echo replies to reach the protected network even if no ICMP echo request was initiated from the protected network.

Therefore we can use ICMP echo replies, and hope they will get routed through the firewall, inside the protected network. The last hop router, in many cases an internal router, will issue an ICMP host unreachable error message for each IP address it cannot deliver the ICMP echo reply to.

It will not only reveal the non-existence of the targeted IP address, but the presence of an internal router.

```
00:15:18 prober> Targeted_IP_Address: icmp: echo reply
00:15:19 router> prober: icmp: host unreachable
```

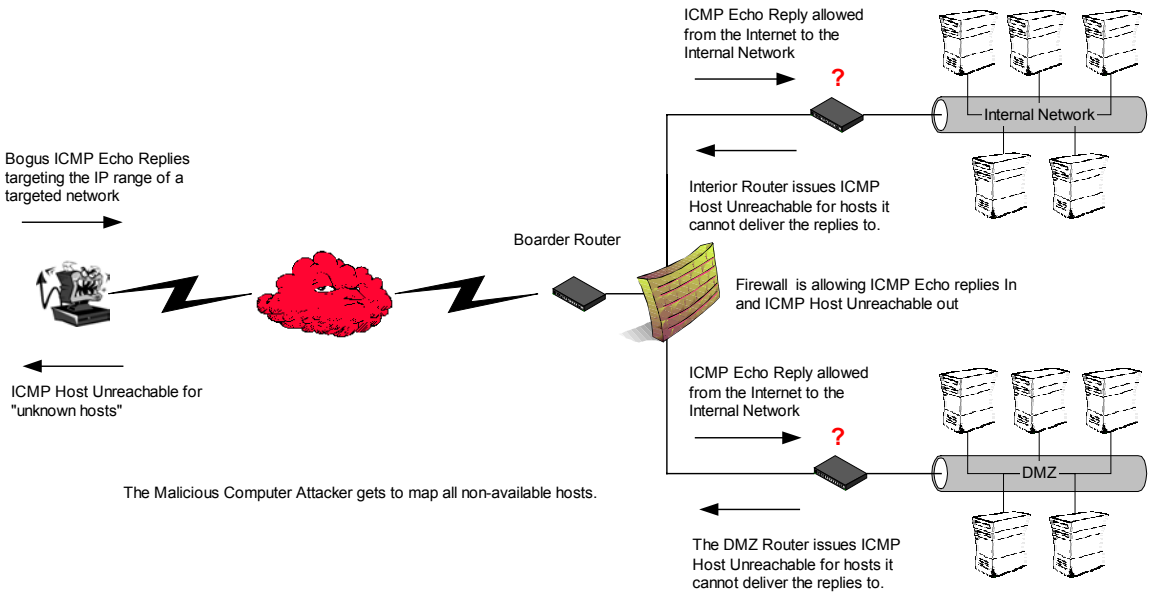


Figure 23: Inverse Mapping Using ICMP Echo Replies

5.2 Inverse Mapping Using Other Protocols

The technique of inverse mapping will work with other protocols as the stimulus as well. It will produce the same results since the destination IP address will still be unreachable. The last hop router's Arp requests will still be not answered, and therefore the router will issue an ICMP Host Unreachable error message (regardless of the underlying protocol used) back to the offending packet's source IP address.

Using ICMP as the underlying protocol might be more beneficial, especially ICMP echo replies, when a filtering device is present and protecting a targeted network.

5.3 Patterns we might see

This type of scan will produce a number of patterns. Not always, when we will see a router issuing an ICMP host unreachable error message it will be because some one meant to use the inverse mapping technique.

Lets look at our first example:

```
Router_IP > The_Same_IP : icmp: host Host_A unreachable
Router_IP > The_Same_IP : icmp: host Host_D unreachable
Router_IP > The_Same_IP : icmp: host Host_G unreachable
...
Router_IP > The_Same_IP : icmp: host Host_N unreachable
...
```

The same host is being used to scan an entire IP range of a targeted network. Some of the Hosts the malicious computer attacker has tried to reach were not reachable. Still, the malicious computer attacker gets an idea about what is not reachable. Sometimes these results are the only indication that the malicious computer attacker will have about the presence of Hosts in a targeted network.

Lets look at the next example:

```
18:12:21.901256 Router_IP > 192.168.46.45: icmp: host x.x.x.12
unreachable
18:12:33.676136 Router_IP > 192.168.59.63: icmp: host x.x.x.12
unreachable
18:12:33.676218 Router_IP > 192.168.59.63: icmp: host x.x.x.12
unreachable
18:13:27.084221 Router_IP > 192.168.114.37: icmp: host x.x.x.12
unreachable
18:13:45.559706 Router_IP > 192.168.22.91: icmp: host x.x.x.12
unreachable
18:13:45.559856 Router_IP > 192.168.22.91: icmp: host x.x.x.12
unreachable
18:13:48.413514 Router_IP > 192.168.250.254: icmp: host x.x.x.12
unreachable
18:13:48.413681 Router_IP > 192.168.250.254: icmp: host x.x.x.12
unreachable
18:14:31.313495 Router_IP > 192.168.247.186: icmp: host x.x.x.12
unreachable
18:14:31.313624 Router_IP > 192.168.247.186: icmp: host x.x.x.12
unreachable
18:15:32.884187 Router_IP > 192.168.12.213: icmp: host x.x.x.12
unreachable
...
```

With this example different Hosts fail to reach the x.x.x.12 IP address. The last hop router is sending them all an ICMP Host Unreachable error message.

How come different IP addresses are seeking the same host on such a short notice?

Probably what we are seeing here is a decoy scan. A decoy scan is a type of scan, which involves multiple IP addresses, which are fed to the network-scanning tool as decoys. The real IP address of the malicious computer attacker (or a machine he controls) will be among those.

The defending side will have difficulties in realizing what was the real IP address the malicious computer attacker was using among all the IP addresses probing the network.

With our example the IP address is reported, to all seeking IP addresses, to be unreachable. The last hop router is trying to deliver the packets but fails to get an answer for his arp requests.

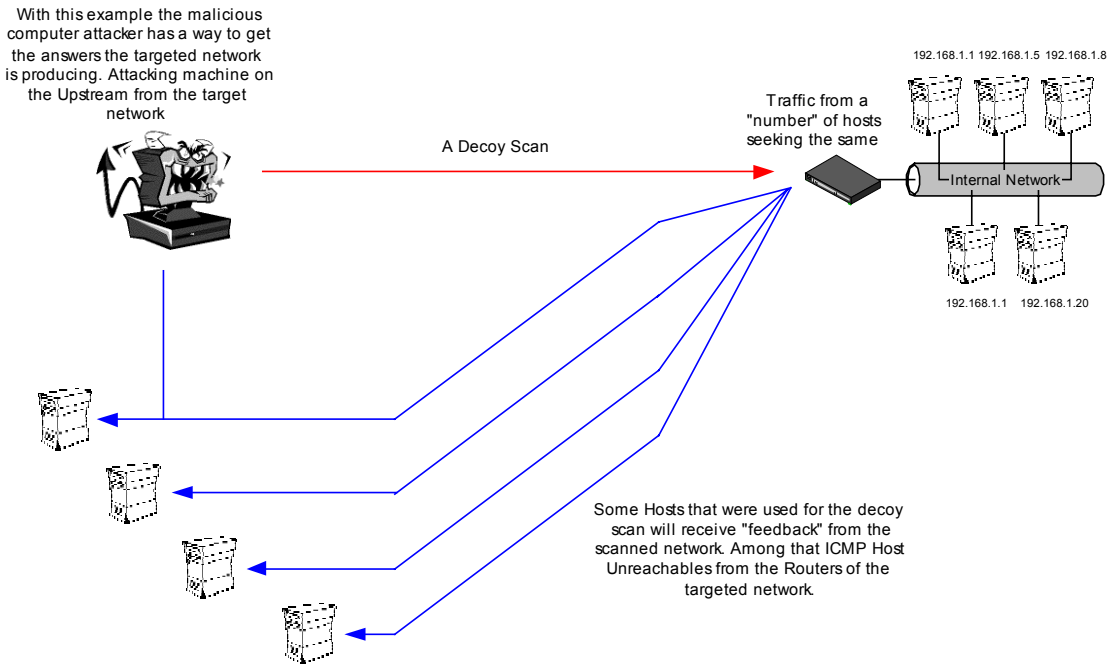


Figure 24: A Decoy Scan Example

Countermeasure: Block outgoing ICMP Time Exceeded in Transit and ICMP Host Unreachable error messages from your protected network to the Internet. Use a real dynamic/statefull inspection firewall.

6.0 Using traceroute to Map a Network Topology

Traceroute is a network debugging utility, which attempts to map all networking devices and hosts on a route to a certain destination host/machine.

The *NIX version of the program sends UDP (by default) or ICMP Echo Request[38] datagrams in sets of three, to a certain destination host. The first three datagram's to be sent have an IP Time-to-Live field value equal to one. The program relies on the fact that the IP Time to Live field value is decreased at each point that the IP header is being processed. A router should decrement the TTL field value just before forwarding the datagram to another router/gateway. If a router discovers that the Time-To-Live field value in an IP header of a datagram he process equals zero (or less) he would discard the datagram and generate an ICMP Time Exceeded in transit error message back to the offending packet's source IP address.

This is when a successful round is completed and another set of three datagrams is sent, this time with a Time-to-Live field value greater by one than the last set.

The originating host would know at which router the datagram triggered the ICMP error message since it receives this information with the ICMP Time to Live Exceeded in Transit error message (Source IP address of the ICMP error message would be the IP address of the router/gateway; With the offending packets data carried with the error message we will have additional information that will bound this ICMP error message to our issued traceroute command).

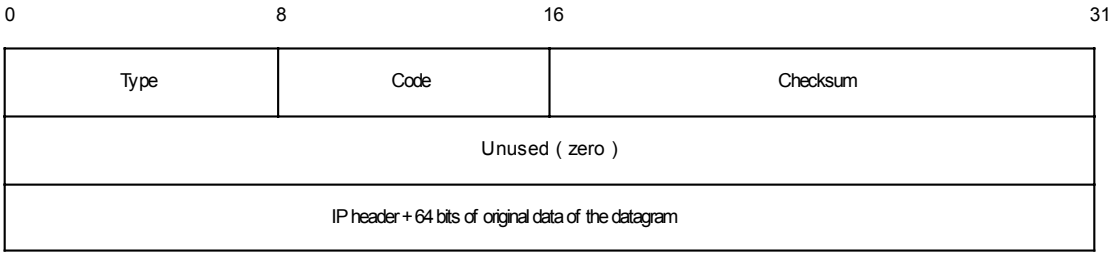


Figure 25: ICMP Time Exceeded message format

We increase the IP Time to Live field value, starting from one, for each successful round (a round is finished when an ICMP Time Exceeded in Transit error message is received) until we receive an ICMP Port Unreachable error message (or ICMP Echo Reply if we are using ICMP Echo request datagrams) from the destined machine. This way we map every router/gateway/host along the path to our destination.

By default, when sending UDP datagrams we use a destination port, which is probably, not used by the destination host so the UDP datagrams will trigger ICMP Port Unreachable error messages back from the destined machine, when reaching it. The destination port will be increased with each probe sent.

We get ICMP responses provided there is no prohibitive filtering or any packet loss.

[38] Microsoft Windows NT and Microsoft Windows 2000 are using the 'tracert' utility, which uses ICMP Echo Request datagrams as its default.

The output we see with the traceroute utility is a line showing the Time-To-Live field value, the IP address of the host/gateway, and the round trip time of each probe. If we do not get a response back within 5 seconds an asterisk ("*") is printed, which represents no answer.

The next example was produced with the '`tracert`' utility with Microsoft Windows 2000:

```
C:\>tracert www.sys-security.com
Tracing route to www.sys-security.com [216.230.199.48]
over a maximum of 16 hops:
  1   100 ms   100 ms   120 ms  Haifa-mng-1 [213.8.12.7]
  2    90 ms    90 ms    90 ms  ge037.herndon1.us.telia.net [205.164.141.1]
  3   120 ms   151 ms   200 ms  213.8.8.5
  4   441 ms   450 ms   451 ms  500.Serial3-5.GW3.NYC6.ALTER.NET [157.130.253.69]
  5   440 ms   451 ms   451 ms  521.ATM2-0.XR2.NYC4.ALTER.NET [152.63.24.38]
  6   912 ms   460 ms   461 ms  188.ATM3-0.TR2.NYC1.ALTER.NET [146.188.179.38]
  7   471 ms   480 ms   471 ms  104.at-5-1-0.TR2.CHI4.ALTER.NET [146.188.136.153]
  8   470 ms   471 ms   471 ms  198.at-2-0-0.XR2.CHI2.ALTER.NET [152.63.64.229]
  9   480 ms   471 ms   471 ms  0.so-2-1-0.XL2.CHI2.ALTER.NET [152.63.67.133]
 10   471 ms   471 ms   470 ms  POS6/0.GW2.CHI2.ALTER.NET [152.63.64.145]
 11   471 ms   481 ms   470 ms  siteprotect.customer.alter.net [157.130.119.50]
 12   481 ms   490 ms   481 ms  216.230.199.48
Trace complete.
C:\>
```

6.1 When A Firewall Protects a Network

In the next scenario a firewall is protecting a targeted network. The only traffic allowed is DNS queries aimed at the targeted network's DNS server, using UDP port 53.

With this scenario, performing a regular traceroute aimed at the DNS machine's IP address will result with traces stopped at the entrance point to the network, hence the Firewall. This is since the destination UDP port used in the queries is being blocked by the Firewall[39].

```
zuul:~>traceroute 10.0.0.10
traceroute to 10.0.0.10 (10.0.0.10), 30 hops max, 40 byte
packets
1 10.0.0.1 (10.0.0.1) 0.540 ms 0.394 ms 0.397 ms
2 10.0.0.2 (10.0.0.2) 2.455 ms 2.479 ms 2.512 ms
3 10.0.0.3 (10.0.0.3) 4.812 ms 4.780 ms 4.747 ms
4 10.0.0.4 (10.0.0.4) 5.010 ms 4.903 ms 4.980 ms
5 10.0.0.5 (10.0.0.5) 5.520 ms 5.809 ms 6.061 ms
6 10.0.0.6 (10.0.0.6) 9.584 ms 21.754 ms 20.530 ms
7 10.0.0.7 (10.0.0.7) 89.889 ms 79.719 ms 85.918 ms
8 10.0.0.8 (10.0.0.8) 92.605 ms 80.361 ms 94.336 ms
9 * * *
10 * * *
```

If we wish to reach the DNS server we need to set the UDP port number with our probes to 53. The traceroute utility increases the port number each time it sends a UDP datagram, therefore we need to calculate the port number to start with, so when a datagram will be processed by the Firewall[40] and will be examined, it will have the appropriate port and relevant information needed to comply with the Access Control List which the Firewall enforces on the targeted network. We can use a simple equation to calculate the starting port:

[39] All examples taken from "A Traceroute-Like Analysis of IP Packet Responses to Determine Gateway Access Control Lists" by David Goldsmith and Michael Shiffman. No real examples were provided because of legal issues.

[40] A firewall should not elicit any reply or ICMP error messages for any traffic destined directly at the Firewall.

(Target port – (number of hops * number of probes)) -1

The number of hops (gateways) from the probing host to the firewall is taken from our earlier traceroute. We use three probes for every query with the same IP Time-to-Live field value; each one of them uses a different destination port number.

```
zuul:~>traceroute -p28 10.0.0.10
traceroute to 10.0.0.10 (10.0.0.10), 30 hops max, 40 byte packets
1 10.0.0.1 (10.0.0.1) 0.501 ms 0.399 ms 0.395 ms
2 10.0.0.2 (10.0.0.2) 2.433 ms 2.940 ms 2.481 ms
3 10.0.0.3 (10.0.0.3) 4.790 ms 4.830 ms 4.885 ms
4 10.0.0.4 (10.0.0.4) 5.196 ms 5.127 ms 4.733 ms
5 10.0.0.5 (10.0.0.5) 5.650 ms 5.551 ms 6.165 ms
6 10.0.0.6 (10.0.0.6) 7.820 ms 20.554 ms 19.525 ms
7 10.0.0.7 (10.0.0.7) 88.552 ms 90.006 ms 93.447 ms
8 10.0.0.8 (10.0.0.8) 92.009 ms 94.855 ms 88.122 ms
9 10.0.0.9 (10.0.0.9) 101.163 ms * *
10 * * *
```

We face another problem.

After the datagram that have used UDP port 53 passed the ACL of the firewall and listed the outer leg of the firewall itself as the next hop, the next UDP datagram sent would be with a different port number - Than again it would be blocked by the firewall's rule base.

A modification to the traceroute utility has been made by Michael Shiffman[41] in order to stop the port increasement. A side effect from using the traceroute utility with a fixed port number will be not receiving an ICMP Port Unreachable error message back from a destination host. This is due to the fact that the port we will use with our queries might be in listening state on the targeted host.

```
zuul:~>traceroute -S –p53 10.0.0.15
traceroute to 10.0.0.15 (10.0.0.15), 30 hops max, 40 byte
packets
1 10.0.0.1 (10.0.0.1) 0.516 ms 0.396 ms 0.390 ms
2 10.0.0.2 (10.0.0.2) 2.516 ms 2.476 ms 2.431 ms
3 10.0.0.3 (10.0.0.3) 5.060 ms 4.848 ms 4.721 ms
4 10.0.0.4 (10.0.0.4) 5.019 ms 4.694 ms 4.973 ms
5 10.0.0.5 (10.0.0.5) 6.097 ms 5.856 ms 6.002 ms
6 10.0.0.6 (10.0.0.6) 19.257 ms 9.002 ms 21.797 ms
7 10.0.0.7 (10.0.0.7) 84.753 ms * *
8 10.0.0.8 (10.0.0.8) 96.864 ms 98.006 ms 95.491 ms
9 10.0.0.9 (10.0.0.9) 94.300 ms * 96.549 ms
10 10.0.0.10 (10.0.0.10) 101.257 ms 107.164 ms 103.318 ms
11 10.0.0.11 (10.0.0.11) 102.847 ms 110.158 ms *
12 10.0.0.12 (10.0.0.12) 192.196 ms 185.265 ms *
13 10.0.0.13 (10.0.0.13) 168.151 ms 183.238 ms 183.458 ms
14 10.0.0.14 (10.0.0.14) 218.972 ms 209.388 ms 195.686 ms
15 10.0.0.15 (10.0.0.15) 236.102 ms 237.208 ms 230.185 ms
```

Countermeasure

You need to configure your firewall and border routers correctly:

- Configure your border routers not to generate ICMP Time to Live exceeded in transit error messages.
- Configure your border routers not to answer any traffic aimed directly at the routers, unless the traffic is about routing information.

[41] http://www.packetfactory.net

- Do not allow any traffic destined the firewall.
- Do not allow any ICMP error message generated by the firewall from reaching the Internet (and from reaching internal segments).
- Block packets coming with low IP Time to Live field values from entering your network. Some firewalls have this ability already implemented. Please consult your firewall manufacture.
- Disallow any ICMP Time to Live exceeded in transit error message coming from a protected network destined the Internet.

7.0 The usage of ICMP in Active Operating System Fingerprinting Process

We use Finger Printing techniques in order to detect the underlying operating system a targeted host is using.

This piece of information is one of few pieces of information a malicious computer attacker will try to have in deciding if to lunch an attack attempt on a targeted host.

Other pieces of information will be:

- A target, a host detected using once of the host detection methods.
- Services, which are running on the targeted host (open ports). This will be done with one of the Port Scanning methods.
- Operating System being used on the targeted host.

Combining the information will allow the malicious computer attacker to identify if the targeted host is vulnerable to a certain exploit aimed at a certain service version running on a certain operating system.

In this section I have outlined the active operating system fingerprinting methods using the ICMP protocol. Nearly all methods were discovered during this research.

What makes the Active Fingerprinting methods, which are using the ICMP protocol unique comparing to other Active OS Fingerprinting methods?
As we will learn, using active OS fingerprinting methods with the ICMP protocol requires less traffic initiation from the malicious computer attacker's machine to a target host, in order to determine its underlying operating system.

With some methods only one datagram is required to determine the underlying operating system.

7.1 Using Regular ICMP Query Messages

7.1.1 The "Who answer what?" approach

The question "Which operating system answer for what kind of ICMP Query messages?" help us identify certain groups of operating systems.

For example, Linux and *BSD based operating systems with default out-of-the-box installation answer for ICMP Echo requests and for ICMP Timestamp Requests. Until Microsoft Windows 2000 family of operating systems has been released it was a unique combination for these two groups of operating systems. Since the Microsoft Windows 2000 operating system family mimics the same behavior (yes mimic), it is no longer feasible to make this particular distinction.

Microsoft might have been thinking that this way of behavior might hide Microsoft windows 2000 machines in the haze. As we will see with the examples given in this research paper they have much more to learn.

The thing is there is no clear distinction between one operating system to another based on this method. We can only group operating systems together and try other methodologies in order to divide those groups a bit more[42].

For the complete mapping of the operating systems I have queried for this research please see "*Appendix B: Mapping Operating Systems for answering/ discarding ICMP query message types*", and "*Appendix D: ICMP Query Message Types aimed at a Broadcast Address*".

7.1.1.1 Identifying Operating Systems according to their replies for non-ECHO ICMP query requests aimed at the broadcast address

If IP directed broadcasts are not blocked, than we can identify answering hosts quite easily.

The first step will be sending an ICMP Timestamp request aimed at the broadcast address of a targeted network. The operating systems that will answer will include Sun Solaris, HP-UX 10.20, and Linux based on Kernel version 2.2.x. We can further identify these operating systems by sending an ICMP Information request aimed at the broadcast address of the targeted network. HP-UX 10.20 will answer the query while Sun Solaris and Linux will not. To distinguish between these two we will send an ICMP Address Mask request to the IP addresses that did not answer in the previous step. Sun Solaris will reply to the query while Linux machines based on Kernel 2.2.x will not.

[42] Note: If the PMTU Discovery process using ICMP Echo requests is enables with HP-UX 10.30 & 11.0x operating systems than our simple query will trigger a "retaliation" from those machines, enabling us to identify them very easily.

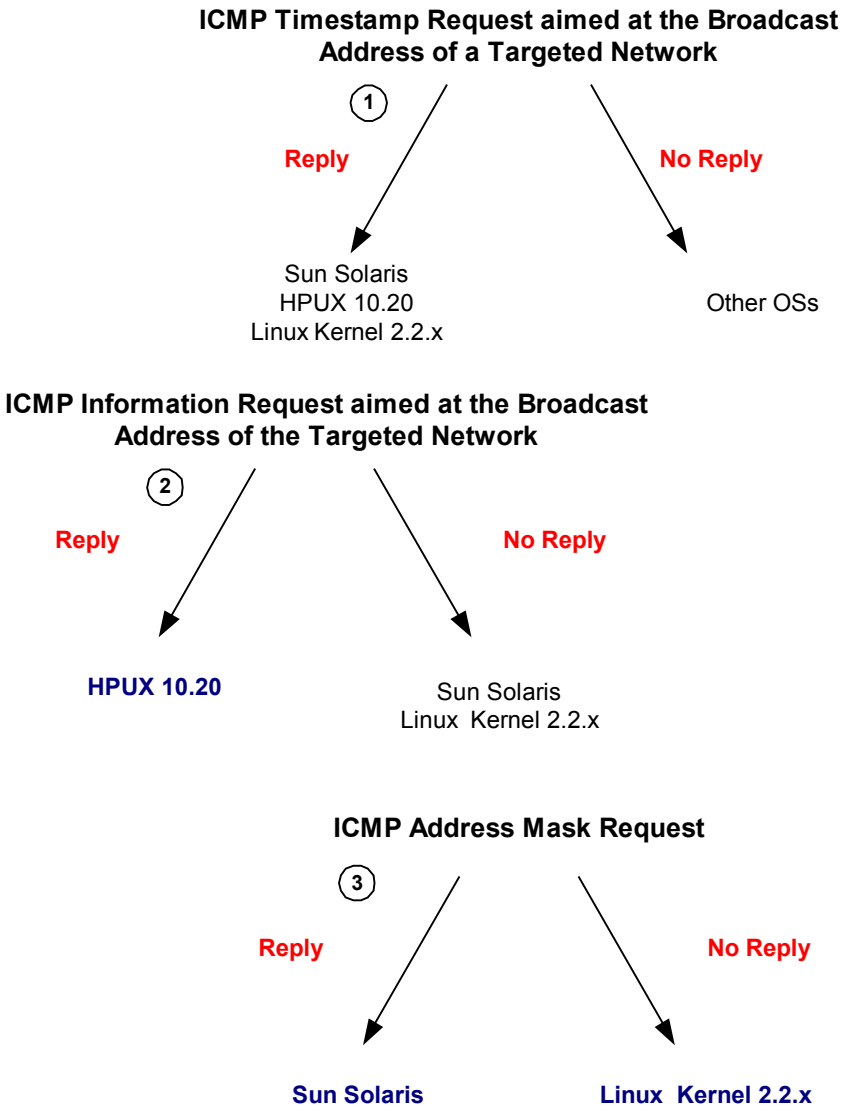


Diagram 1: Finger Printing Using non-ECHO ICMP Query Types aimed at the Broadcast Address of an Attacked Network

Examining the IP ID field value(s)

RFC 791 gives a description about the IP Identification field.

The identification field value is used to uniquely identify the fragments of a particular datagram. Fragments of a particular datagram are assembled if they have the same source, destination, protocol, and Identifier. The identifier is being chosen to be unique for this "this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet"[43].

[43] RFC 791: Internet Protocol. http://www.ietf.org/rfc/rfc0791.txt.

The IP identifier field can have 65,536 different values. It is important for an operating system to have some sort of a mechanism in order to control the identification numbers correctly.

Since every operating system should have its own mechanism in order to deal with this field numbering we might find some patterns different from one operating system to another.

7.1.2 Identifying Kernel 2.4.x Linux based machines using the IP ID field with ICMP datagrams

While examining Linux Kernel 2.4.x, I have encounter a rather simple operating system fingerprinting method using the ICMP protocol targeting machines based on Linux Kernel 2.4.x.

In the next example the IP address 192.168.1.1 is a Linux machine running Kernel 2.2.14, the IP address 192.168.1.10 is a Linux machine running Kernel 2.4.2. We are using the '`ping`' utility to generate ICMP Echo requests:

```
17:23:03.623486 eth0 > 192.168.1.1 > 192.168.1.10: icmp: echo request
(ttl 64, id 68)
                  4500 0054 0044 0000 4001 f709 c0a8 0101
                  c0a8 010a 0800 0600 9808 0000 c734 d93c
                  c582 0900 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
17:23:03.623779 eth0 < 192.168.1.10 > 192.168.1.1: icmp: echo reply
(DF) (ttl 255, id 0)
                  4500 0054 0000 4000 ff01 f84c c0a8 010a
                  c0a8 0101 0000 0e00 9808 0000 c734 d93c
                  c582 0900 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
17:23:04.622911 eth0 > 192.168.1.1 > 192.168.1.10: icmp: echo request
(ttl 64, id 69)
                  4500 0054 0045 0000 4001 f708 c0a8 0101
                  c0a8 010a 0800 ef01 9808 0100 c834 d93c
                  da80 0900 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
17:23:04.623200 eth0 < 192.168.1.10 > 192.168.1.1: icmp: echo reply
(DF) (ttl 255, id 0)
                  4500 0054 0000 4000 ff01 f84c c0a8 010a
                  c0a8 0101 0000 f701 9808 0100 c834 d93c
                  da80 0900 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
```

The IP ID field value with the ICMP Echo replies, generated by the Kernel 2.4.x based machine, is zero (0) and not changing.

I have examined this behavior with the ICMP Timestamp mechanism as well. This time I have used the '`sing`' utility to generate the ICMP Timestamp requests (this is why the IP ID field value in the requests equal to 13170):

```
17:22:10.119231 eth0 > 192.168.1.1 > 192.168.1.10: icmp: time stamp
request (ttl 255, id 13170)
                  4500 0028 3372 0000 ff01 0507 c0a8 0101
                  c0a8 010a 0d00 041c 9508 0000 0315 56c6
                  0000 0000 0000 0000
17:22:10.119431 eth0 < 192.168.1.10 > 192.168.1.1: icmp: time stamp
reply (DF) (ttl 255, id 0)
                  4500 0028 0000 4000 ff01 f878 c0a8 010a
                  c0a8 0101 0e00 42b5 9508 0000 0315 56c6
                  03b1 5c82 03b1 5c82 0000 0000 0000
17:22:11.112908 eth0 > 192.168.1.1 > 192.168.1.10: icmp: time stamp
request (ttl 255, id 13170)
                  4500 0028 3372 0000 ff01 0507 c0a8 0101
                  c0a8 010a 0d00 ff39 9508 0100 0315 5aa8
                  0000 0000 0000 0000
17:22:11.113151 eth0 < 192.168.1.10 > 192.168.1.1: icmp: time stamp
reply (DF) (ttl 255, id 0)
                  4500 0028 0000 4000 ff01 f878 c0a8 010a
                  c0a8 0101 0e00 35fb 9508 0100 0315 5aa8
                  03b1 606e 03b1 606e d039 0100 d039
```

Again the IP ID field value with the ICMP Timestamp replies equals to zero (0) and not changing.

Even when sending ICMP Echo requests from the machine running Linux Kernel 2.4.2 the IP ID field value is fixed and equal to zero (0):

```
05/08/01-15:09:59.573546 172.18.2.201 -> 172.18.2.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:8741   Seq:0  ECHO
17 E2 F7 3A 62 D5 08 00 08 09 0A 0B 0C 0D 0E 0F  ...:b...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567


05/08/01-15:09:59.573546 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:12812 IpLen:20 DgmLen:84
Type:0  Code:0  ID:8741  Seq:0  ECHO REPLY
17 E2 F7 3A 62 D5 08 00 08 09 0A 0B 0C 0D 0E 0F  ...:b...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567


05/08/01-15:10:00.573546 172.18.2.201 -> 172.18.2.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8  Code:0  ID:8741   Seq:256  ECHO
18 E2 F7 3A 1F C3 08 00 08 09 0A 0B 0C 0D 0E 0F  ...:............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

```
05/08/01-15:10:00.573546 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:12813 IpLen:20 DgmLen:84
Type:0  Code:0  ID:8741  Seq:256  ECHO REPLY
18 E2 F7 3A 1F C3 08 00 08 09 0A 0B 0C 0D 0E 0F  ...:............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                         01234567
```

I have downloaded and compiled Kernel 2.4.4 (the latest in the 2.4.x series as of this writing), and observed the same behavior.

This operating system fingerprinting method can be used passively and actively.

7.1.3 Fun with IP Identification Field Values

Identifying Older Microsoft Based OSs

Since every operating system should have its own mechanism in order to deal with the IP identification field numbering we might find some patterns different from one operating system to another.

The Gap between one IP ID field value to the next

With the implementation in many operating systems, the Kernel is increasing the IP ID field value by 1, from one packet to the next.

However, there are operating systems that will increase the value of the IP ID field value with a value different than 1, from one packet to the next.

In the next example I have sent two ICMP Echo requests from a Windows NT 4 Server SP6a based machine targeting a Linux machine based on Kernel 2.2.14:

```
08/10-16:55:06.638539 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:28416
ID:256   Seq:768  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:06.638592 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:1452
ID:256   Seq:768  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:07.639784 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:28672
ID:256   Seq:1024  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:07.639841 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:1453
ID:256   Seq:1024  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
```

```
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

The first ICMP Echo request sent from the Microsoft NT 4 based machine was sent with IP ID field value of 28416. The second ICMP Echo request was sent with IP ID field value of 28672. Simple calculation will show a gap of 256 between the IP ID field values.

Looking at the replies the Linux based machine produced, we see a gap of 1 between one IP ID to the next.

Other OSs that act the same
The other operating systems that act as the Microsoft NT 4 SP6a based machine are the older Microsoft based operating systems. They include - Windows 95, Windows 98, Windows 98 SE, Windows NT 4 family (regardless of the service pack installed).

With newer versions of their operating systems (MS Windows ME, MS Windows 2000 family), Microsoft has changed this behavior, and now acts as most operating systems do.

How Can We Use This?
We can use this information as another parameter for Active OS fingerprinting and for Passive OS fingerprinting.

One example might be when we need another parameter to differentiate between a Windows NT 4 based machine to a Windows 2000 based machine.

In The Real World
In the real world when we wish to use this information for fingerprinting operating systems we will see something a bit different that we should be aware of. Since the machines we try to fingerprint are hosts available on the Internet they might communicate with other hosts on the Internet while we query them. Therefore the gap we will have from one IP ID field value to the next might be higher than 256 (in the older MS based OSs case). With the older implementations of Microsoft based operating systems identifying these OSs is quite simple. We will extract the first IP ID field value from the second IP ID field value and divide the result with 256. The result should be a complete number.

With the operating systems that use a gap of 1 between one IP ID field value to the next, we might have a gap a bit higher than 1, usually between 2-8 (but it can be more than that as well).

In the next example a Microsoft ME based machine sent two ICMP Echo requests targeting a Linux based on kernel 2.2.14 machine. The gap between the first IP ID field value to the next is 5 with the Linux machine:

```
08/10-16:49:45.633417 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:134
ID:768   Seq:256  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:49:45.633465 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:810
ID:768   Seq:256  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
```

```
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:49:46.635971 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:135
ID:768   Seq:512  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:49:46.636018 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:815
ID:768   Seq:512  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

7.1.4 The DF Bit Playground

Identifying Sun Solaris, HP-UX 10.30, 11.0x, Linux With Kernel 2.4.x, and AIX 4.3.x based machines

RFC 791 defines a three bits field used for various control flags in the IP Header.

Bit 0 is the reserved flag, and must be zero.

Bit 1, is called the Don't Fragment flag, and can have two values. A value of zero (not set) is equivalent to May Fragment, and a value of one is equivalent to Don't Fragment. If this flag is set than the fragmentation of this packet at the IP level is not permitted, otherwise it is.

Bit 2, is called the More Fragments bit. It can have two values. A value of zero is equivalent to (this is the) Last Fragment, and a value of 1 is equivalent to More Fragments (are coming).

The next field in the IP header is the Fragment Offset field, which identifies the fragment location relative to the beginning of the original un-fragmented datagram (RFC 791, bottom of page 23).

A close examination of the ICMP Query replies would reveal that some operating systems will set the DF bit with their replies.

In the next example I have sent an ICMP Echo request to www.openbsd.org. The web site is run on a Sun Solaris platform, since it is being hosted:

```
[root@godfather /]# sing -echo -c 2 www.openbsd.org
SINGing to www.openbsd.org (129.128.5.191): 16 data bytes
16 bytes from 129.128.5.191: seq=0 DF! ttl=238 TOS=0 time=325.439 ms
16 bytes from 129.128.5.191: seq=1 DF! ttl=238 TOS=0 time=439.743 ms

--- www.openbsd.org sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 325.439/382.591/439.743 ms
```

This is the `snort` trace:

```
05/20/01-17:16:45.137465 172.18.2.201 -> 129.128.5.191
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:36
Type:8  Code:0  ID:62725   Seq:0  ECHO
```

```
CD D1 07 3B 00 27 02 00                          ...;.'..


05/20/01-17:16:45.457465 129.128.5.191 -> 172.18.2.201
ICMP TTL:238 TOS:0x0 ID:23189 IpLen:20 DgmLen:36 DF
Type:0  Code:0  ID:62725  Seq:0  ECHO REPLY
CD D1 07 3B 00 27 02 00
```

The DF bit is not only set with ICMP Echo replies, it is also being set on all other types of ICMP Query replies the underlying operating system is supporting.

Other operating systems, which set the DF bit with their ICMP Query replies by default, are Linux based on Kernel 2.4.x, HPUX 10.30 & 11.0x, and AIX 4.3.x.

As we have seen with Linux in the previous section, the IP ID field value with the ICMP Query replies will be equal to zero. This will enable us to differentiate between Linux Kernel 2.4.x based machines to the rest of the operating systems producing this behavior.

```
[root@godfather /root]# sing -echo -c 2 172.18.2.201
SINGing to 172.18.2.201 (172.18.2.201): 16 data bytes
16 bytes from 172.18.2.201: seq=0 DF! ttl=255 TOS=0 time=2.349 ms
16 bytes from 172.18.2.201: seq=1 DF! ttl=255 TOS=0 time=2.207 ms

--- 172.18.2.201 sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.207/2.278/2.349 ms
```

The `snort` trace:

```
05/20/01-17:19:51.097465 172.18.2.201 -> 172.18.2.201
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:36
Type:8  Code:0  ID:64005   Seq:0  ECHO
87 D2 07 3B B3 9C 01 00                          ...;....


05/20/01-17:19:51.097465 172.18.2.201 -> 172.18.2.201
ICMP TTL:255 TOS:0x0 ID:0 IpLen:20 DgmLen:36 DF
Type:0  Code:0  ID:64005  Seq:0  ECHO REPLY
87 D2 07 3B B3 9C 01 00                          ...;...
```

With HPUX 10.30 & 11.0x and with AIX 4.3.x we will sometimes encounter a slightly different behavioral pattern.

7.1.4.1 HP-UX 10.30 / 11.x & AIX 4.3.x Path MTU Discovery Proccess Using ICMP Echo Requests

HP claims to have a proprietary method in order to determine the Path MTU with HP-UX v10.30, and HP-UX v11.0x using ICMP Echo requests. This method is enabled be default. AIX 4.3.x does exactly the same.

The next trace will help to understand the process taken by HPUX 10.30 & 11.0x and AIX 4.3.x.

With this example I have sent an ICMP Echo request targeting an HP-UX 11.0 based machine. My IP address is represented by *y.y.y.y*, the target's IP address is represented by *x.x.x.x*:

```
00:27:56.884147 ppp0 > y.y.y.y > x.x.x.x: icmp: echo request (ttl 255,
id 13170)
                         4500 0024 3372 0000 ff01 7c51 yyyy yyyy
                         xxxx xxxx 0800 5238 6d04 0000 dce5 c339
                         8b7d 0d00
00:27:57.165620 ppp0 < x.x.x.x > y.y.y.y : icmp: echo reply (ttl 236,
id 41986)
                         4500 0024 a402 0000 ec01 1ec1 xxxx xxxx
                         yyyy yyyy 0000 5a38 6d04 0000 dce5 c339
                         8b7d 0d00
```

The first pair of ICMP Echo request and ICMP Echo reply was pretty usual. My Linux based machine sent an ICMP Echo request and received an ICMP Echo reply from the targeted HPUX 11.0 based machine. One notable detail – *the DF bit was not set in the ICMP Echo reply*.

Than something that was not expectable has happened:

```
00:27:57.435620 ppp0 < x.x.x.x > y.y.y.y : icmp: echo request (DF) (ttl
236, id 41985)
                         4500 05dc a401 4000 ec01 d909 xxxx xxxx
                         yyyy yyyy 0800 7e52 9abc def0 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

00:27:57.435672 ppp0 > y.y.y.y > x.x.x.x: **icmp: echo reply** (ttl 255, id 53)

```
4500 05dc 0035 0000 ff01 a9d6 yyyy yyyy
xxxx xxxx 0000 8652 9abc def0 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

The HPUX 11.0 based machine I have queried pinged me back!

The ICMP Echo request datagram size was 1500 bytes long. It was the maximum transfer unit my Internet Connection was allowed to process (dialup using PPP). The request was sent with the DF bit set.

The process will begin when a host will query an HPUX 11.x based machine. The HPUX based machine will send an ICMP Echo request to the querying host with a datagram size that equal to its physical layer's MTU. The data field with the request will be all zeroes. Any router along the way, trying to fragment the request because the MTU of a destined network is smaller than the datagram's size, will fail. The router will send an ICMP Fragmentation Needed but the DF bit was set error message back to the offending packet's source IP address (in this case the HPUX 11.x based machine). It will trigger the HPUX's PMTU discovery algorithm to send a smaller sized ICMP Echo request datagram this time. The process will end when the HPUX 11.x based machine will receive an ICMP Echo reply for one of the ICMP Echo requests initiated by the PMTU discovery algorithm to the querying host. Than the Path MTU between the two ends is discovered, and the process will end.

The following ICMP Echo request was sent from my machine to the targeted HP-UX 11.0x based machine just milliseconds after my reply to the HP-UX's query was sent. This time the DF bit was set with the ICMP Echo reply I have received.

```
00:27:57.885662 ppp0 > y.y.y.y > x.x.x.x : icmp: echo request (ttl 255,
id 13170)
                  4500 0024 3372 0000 ff01 7c51 yyyy yyyy
                  xxxx xxxx 0800 5832 6d04 0100 dde5 c339
                  8383 0d00
00:27:58.155627 ppp0 < x.x.x.x > y.y.y.y : icmp: echo reply (DF) (ttl
236, id 41987)
                  4500 0024 a403 4000 ec01 debf xxxx xxxx
                  yyyy yyyy 0000 6032 6d04 0100 dde5 c339
                  8383 0d00
```

Rather than sending future datagrams from the HPUX 11.x based machine to my machine that will have to be fragmented somewhere along the way, it is more beneficial from performance perspective, to fragment the future datagrams on sending.

Setting the DF bit on the following ICMP query replies will help to maintain the PMTU between the two hosts. If for any reason, the PMTU will be decreased. One of the reasons might be that datagrams may take different routes to the destoinsation.

Sending immediately another ICMP query message type to the HP-UX 11.x operating system based machine, will not result in the PMTU discovery process to be repeated. The DF Bit will be set with the ICMP query reply. Expect a threshold to be maintained by the HP-UX 11.x operating systems based machines. When reached, the next time we query this host with any type of communication, the process of determining the PMTU using ICMP Echo requests will be initiated again.

Why this method is bound to failure?

- Some operating systems will be configured not to reply for an ICMP Echo requests.
- This ability can be used for a denial-of-service attack using HPUX 10.30, and/or HPUX 11.x based machines as an amplifier for these attacks. Infact, HP has released a security bulletin dated February 13, 2000 about some issues regarding this PMTU discovery capability with ICMP. The bulletin states that "*Depending upon the amount and nature of the inbound traffic, an HP-UX 10.30/11.00/11.04 system*

can be used to flood a target system with IP packets which could result in a denial of service"[44].

- Easy identification of HP-UX 10.30, 11.x, and AIX 4.3.x based machines.

This unique behavior, when experienced, help us to distinguish between Sun Solaris based machines, HP-UX 11.0x/10.30 based machines, and AIX 4.3.x based machines.

Sun Solaris sets the DF bit with the ICMP query replies the operating system answers for, in order to support its global PMTU discovery process. If a Sun Solaris based machine will receive an ICMP fragmentation needed but the DF bit was set error message for an ICMP query reply it had issued, than the size of the MTU used will be lowered. Since ICMP datagrams are small in size, I do not expect this scenario to happen. There is no active measures with Sun Solaris as far as I know.

The following operating systems where queried and checked for this kind of behavior:
Linux Kernel 2.4 test 2,4,5,6; Linux Kernel 2.2.x; Linux Kernel 2.4.x; FreeBSD 4.0, 3.4; OpenBSD 2.7,2.6; NetBSD 1.4.1,1.4.2; BSDI BSD/OS 4.0,3.1; Solaris 2.6,2.7,2.8; HP-UX 10.20, 11.0x; Compaq Tru64 5.0; Aix 4.1,3.2; Irix 6.5.3, 6.5.8; Ultrix 4.2 – 4.5; OpenVMS v7.1-2; Novel Netware 5.1 SP1, 5.0, 3.12; Microsoft Windows 98/98SE/ME, Microsoft Windows NT WRKS SP6a, Microsoft Windows NT Server SP4, Microsoft Windows 2000 Family.

7.1.4.2 Detection Avoidance

With Sun Solaris and HPUX operating systems we can use a configuration option to order the operating system not to set the DF bit with the ICMP query replies[45].

This will prevent us from using the fingerprinting method I have introduced.

Please pay attention to the details. Turning off this ability might break some other things with your TCP/IP communications, especially with Sun Solaris based machines.

7.1.4.2.1 HPUX

With HPUX 10.30 and 11.x we will have to turn off the Path MTU Discovery process using ICMP query requests.

With HP-UX 10.30, & 11.0[46], one of the ndd command options is the *ip_pmtu_strategy*. The variable settings for this option are either 1 or 2. If this bit value is 2, than the Path MTU Discovery Process is used with ICMP Echo Requests. This is the default value. If this bit value equals 1, than the HPUX based machines will not use the ICMP echo request PMTU discovery strategy, and will not set the DF bit after determining the accurate PMTU.

7.2.4.2.2 Sun Solaris

To turn off *ip_path_mtu_discovery* on a Sun Solaris machine use the following command as root:

```
# ndd -set  /dev/ip  ip_path_mtu_discovery 0
```

[44] HP Security Bulletin - "Security Vulnerability with PMTU strategy (revised)", February 13. 2000.
[45] I do not have any information regarding AIX.
[46] Building a Bastion Host Using HP UX 11, Kevin Stevens, http://people.hp.se/stevesk/bastion11.html.

When an ICMP Echo reply is received from the Sun Solaris queried host the DF bit will not be set:

```
[root@godfather /root]# sing -echo -c 1 IP_Address
SINGing to Host_Address (IP_Address): 16 data bytes
16 bytes from 10.13.57.20: icmp_seq=0 ttl=254 TOS=0 time=1.578 ms

--- Host_Address sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.578/1.578/1.578 ms
```

Beware - With Sun Solaris turning this option off, will turn off the PMTU discovery process with other protocols as well. This is not recommended because of performance issues.

7.2.4.2.3 Linux Kernel 2.4.x

With Linux based on Kernel 2.4.x the DF bit will always be set, this is even if we set the *ip_no_pmtu_disc* parameter to 1.

```
[root@godfather ipv4]# echo 1 > ip_no_pmtu_disc
[root@godfather ipv4]# /etc/rc.d/init.d/network restart
Shutting down interface eth0:                              [  OK  ]
Setting network parameters:                                [  OK  ]
Bringing up interface lo:                                  [  OK  ]
Bringing up interface eth0:                                [  OK  ]
```

The behavior pattern *will not change*:

```
[root@godfather /root]# sing -echo -c 2 172.18.2.201
SINGing to 172.18.2.201 (172.18.2.201): 16 data bytes
16 bytes from 172.18.2.201: seq=0 DF! ttl=255 TOS=0 time=2.315 ms
16 bytes from 172.18.2.201: seq=1 DF! ttl=255 TOS=0 time=2.263 ms

--- 172.18.2.201 sing statistics ---

2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.263/2.289/2.315 ms
```

7.1.5 The IP Time-to-Live Field Value with ICMP

The sender sets the time to live field to a value that represents the maximum time the datagram is allowed to travel on the Internet.

The field value is decreased at each point that the Internet header is being processed. RFC 791 states that this field decreasement reflects the time spent processing the datagram. The field value is measured in units of seconds. The RFC also states that the maximum time to live value can be set to 255 seconds, which equals to 4.25 minutes. The datagram must be discarded if this field value equals zero - before reaching its destination.

Relating to this field as a measure to assess time is a bit misleading. Some routers may process the datagram faster than a second, and some may process the datagram longer than a second.

The real intention is to have an upper bound to the datagrams lifetime, so infinite loops of undelivered datagrams will not jam the Internet.

Having a bound to the datagram's lifetime help us to prevent old duplicates to arrive after a certain time elapsed. So when we retransmit a piece of information which was not previously delivered we can be assured that the older duplicate is already discarded and will not interfere with the process.

The IP TTL field value with ICMP has two separate values, one for ICMP query messages and one for ICMP query replies.

The IP TTL field value helps us identify certain operating systems and groups of operating systems. It also provides us with the simplest means to add another check criteria when we are querying other host(s) or listening to traffic (sniffing).

7.1.5.1 IP TTL Field Value with ICMP Query Replies

We can use the IP TTL field value with ICMP query reply datagrams to identify certain groups of operating systems. The method discussed in this section is a very simple one. We send an ICMP query request message to a host. If we receive a reply, we would be looking at the IP TTL field value in the IP header of the ICMP query reply.

The IP Time-To-Live field value received will not be the original value assigned to this field. The reason is that each router along the path from the targeted host to the prober decreased this field value by one.

We can use two ways to approach this. The first approach will be looking at the IP TTL field values that are ususaly used by operating systems and networking devices. They are 255, 128, 64, 60, and 32. We will use the most close to value, as the original value assigned to the IP TTL field.

The second approach is less accurate than the first one. Since we have already queried the targeted host, querying it again will not be that harmful (well we hope at least). We can use the traceroute program in order to reveal the number of hops between our machine to the targeted host. Adding the number we calculated to the IP TTL field value should give us a good guess about the original IP TTL value assigned to this field.

Why this is only a good guess?

Because the routes taken from the target host to our host and from our host to the target host may be different routes.

The next example demonstrates this behavior. I was using the `ping` and `tracert` utilities with Microsoft Windows 2000:

```
C:\>ping -n 1 www.sys-security.com
Pinging www.sys-security.com [216.230.199.48] with 32 bytes of data:
Reply from 216.230.199.48: bytes=32 time=481ms TTL=238
Ping statistics for 216.230.199.48:
    Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
    Minimum = 481ms, Maximum =  481ms, Average =  481ms
C:\>


C:\>tracert -h 16 www.sys-security.com
Tracing route to www.sys-security.com [216.230.199.48]
over a maximum of 16 hops:
  1   100 ms   100 ms   120 ms  Haifa-mng-1 [213.8.12.7]
  2    90 ms    90 ms    90 ms  ge037.herndon1.us.telia.net [205.164.141.1]
  3   120 ms   151 ms   200 ms  213.8.8.5
  4   441 ms   450 ms   451 ms  500.Serial3-5.GW3.NYC6.ALTER.NET [157.130.253.69]
  5   440 ms   451 ms   451 ms  521.ATM2-0.XR2.NYC4.ALTER.NET [152.63.24.38]
  6   912 ms   460 ms   461 ms  188.ATM3-0.TR2.NYC1.ALTER.NET [146.188.179.38]
  7   471 ms   480 ms   471 ms  104.at-5-1-0.TR2.CHI4.ALTER.NET [146.188.136.153]
  8   470 ms   471 ms   471 ms  198.at-2-0-0.XR2.CHI2.ALTER.NET [152.63.64.229]
  9   480 ms   471 ms   471 ms  0.so-2-1-0.XL2.CHI2.ALTER.NET [152.63.67.133]
 10   471 ms   471 ms   470 ms  POS6/0.GW2.CHI2.ALTER.NET [152.63.64.145]
 11   471 ms   481 ms   470 ms  siteprotect.customer.alter.net [157.130.119.50]
 12   481 ms   490 ms   481 ms  216.230.199.48
Trace complete.
C:\>
```

With the example above, the IP TTL field value of the ICMP Echo reply was equal to 238. Using the –h option I have specified that I wish to use only 16 hops with the queries initiated by the `tracert` utility. The actual number of hops in the path between my machine to the target was only 12 hops.

The ICMP Echo requests sent from my Microsoft Windows 2000 had to travel 12 hops before reaching my web site, while replies from my web site had to go through 17 hops before reaching my machine.

Again, we will have a number close enough to one of the common values used to make a good guess about the original IP TTL field value.

The next table describes the IP TTL field values with ICMP Echo replies for various operating systems. According to the table we can distinguish between certain operating systems:

Operating System	IP TTL on ICMP datagrams - In Reply -
Linux Kernel 2.4	255
LinuxKernel 2.2.14	255
Linux Kernel 2.0.x[47]	64
FreeBSD 4.0	255
FreeBSD 3.4	255
OpenBSD 2.7	255
OpenBSD 2.6	255
NetBSD	255
BSDI BSD/OS 4.0	255
BSDI BSD/OS 3.1	255
Solaris 2.5.1	255
Solaris 2.6	255
Solaris 2.7	255

[47] Stephane Omnes provided information about Linux Kernel 2.0.x.

Operating System	IP TTL on ICMP datagrams - In Reply -
Solaris 2.8	255
HP-UX v10.20	255
HP-UX v11.0	255
Compaq Tru64 v5.0	64
Irix 6.5.3	255
Irix 6.5.8	255
AIX 4.1	255
AIX 3.2	255
ULTRIX 4.2 – 4.5	255
OpenVMS v7.1-2	255
Windows 95	32
Windows 98	128
Windows 98 SE	128
Windows ME	128
Windows NT 4 WRKS SP 3	128
Windows NT 4 WRKS SP 6a	128
Windows NT 4 Server SP4	128
Windows 2000 Family	128

Table 14: IP TTL Field Values in replies from Various Operating Systems

If we would look at the ICMP Echo replies IP TTL field values than we can identify few patterns:

- Nearly all UNIX and UNIX-like operating systems use 255 as their IP TTL field value with ICMP query replies.
- Compaq Tru64 v5.0 and Linux 2.0.x are the exception, using 64 as its IP TTL field value with ICMP query replies.
- Microsoft Windows operating system based machines will be using the value of 128.
- Microsoft Windows 95 is the only Microsoft based operating system to use 32 as its IP TTL field value with ICMP query messages. It makes it unique among all other operating systems.

With the ICMP query replies we have an operating system that is clearly distinguished from the other - Windows 95. Other operating systems are grouped into the ” 64 group” (Linux based Kernel 2.0.x machines & Compaq Tru64 5.0), the “255 group” (UNIX and UNIX-like), and into the “128 group” (Microsoft operating systems).

We are not limited to ICMP Echo replies only. We can use the other ICMP query message types as well, and the results should be the same.

In the next example an I have sent an ICMP Timestamp request to a Linux Kernel 2.2.14 based machine:

```
[root@localhost /root]# sing -tstamp -c 2 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 20 data bytes
20 bytes from y.y.y.y: seq=0 ttl=239 TOS=0 diff=79296
```

```
20 bytes from y.y.y.y: seq=1 ttl=239 TOS=0 diff=79264

--- y.y.y.y sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
[root@localhost /root]#
```

The `snort` trace:

```
05/20/01-17:43:37.027465  x.x.x.x -> y.y.y.y
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:40
Type:13  Code:0  TIMESTAMP REQUEST
13 0A 00 00 03 28 F9 C7 00 00 00 00 00 00 00 00  .....(..........

05/20/01-17:43:37.467465  y.y.y.y -> x.x.x.x
ICMP TTL:239 TOS:0x0 ID:50563 IpLen:20 DgmLen:40
Type:14  Code:0  TIMESTAMP REPLY
13 0A 00 00 03 28 F9 C7 03 2A 2F 87 03 2A 2F 87  .....(...*/..*/.
```

We can use this information with other tests to provide us extra criteria with zero effort.

7.1.5.2 IP TTL Field Value with ICMP ECHO Requests

The examination of the IP TTL field value is not limited to ICMP query replies only. We can learn a lot from ICMP requests aimed at our host(s) as well.

The IP Time-To-Live field value received will not be the original value assigned to this field. The reason is that each router along the path from the querying host to our host(s) will decrease this field value by one.

We will be looking at the IP TTL field values usually used by operating systems and networking devices. They are 255, 128, 64, 60, and 32. We will use the most close to value, as the original value assigned to the IP TTL field.

Using techniques, which will trace the querying host path until his gateway may not work, and may alert the prober that we are aware of his activities[48].

The following table summarizes some operating system's default IP TTL field values with an ICMP query requests:

Operating System	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	255	64
Redhat LINUX 6.2 Kernel 2.2.14	255	64
LINUX Kernel 2.0.x	64	64
FreeBSD 4.0	255	255
FreeBSD 3.4	255	255

[48] See example in the previous section.

Operating System	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
OpenBSD 2.7	255	255
OpenBSD 2.6	255	255
NetBSD	255	
BSDI BSD/OS 4.0	255	
BSDI BSD/OS 3.1	255	
Solaris 2.5.1	255	255
Solaris 2.6	255	255
Solaris 2.7	255	255
Solaris 2.8	255	255
HP-UX v10.20	255	255
HP-UX v11.0	255	
Compaq Tru64 v5.0	64	
Irix 6.5.3	255	
Irix 6.5.8	255	
AIX 4.1	255	
AIX 3.2	255	
ULTRIX 4.2 – 4.5	255	
OpenVMS v7.1-2	255	
Windows 95	32	32
Windows 98	128	32
Windows 98 SE	128	32
Windows ME	128	32
Windows NT 4 WRKS SP 3	128	32
Windows NT 4 WRKS SP 6a	128	32
Windows NT 4 Server SP4	128	32
Windows 2000 Professional	128	128
Windows 2000 Server	128	128

Table 15: IP TTL Field Values in requests from Various Operating Systems

The ICMP query message type used was ICMP Echo request, which is common on all operating systems tested using the `ping` utility.

- The Linux version of `ping` will use 64 as its IP TTL field value with ICMP Echo Requests.
- The `ping` utility with FreeBSD 4.1, 4.0, 3.4; Sun Solaris 2.5.1, 2.6, 2.7, 2.8; OpenBSD 2.6, 2.7, NetBSD and HP UX 10.20 will use 255 as its IP TTL field value with ICMP Echo requests.
- With the `ping` utility with Microsoft Windows 95/98/98SE/ME/NT4 with any service pack, 32 will be used as the IP TTL field value with ICMP Echo requests.
- A Microsoft window 2000 `ping` utility will be using 128 as its IP TTL field value with ICMP Echo requests.

We can distinguish between Linux, Microsoft Windows 2000, the other Microsoft operating systems group, and the “255 group” using this method.

7.1.5.3 Correlating the Information

Using the IP TTL field value with ICMP query messages we can distinguish between Microsoft Windows 95, Microsoft Windows 2000, the other Microsoft Windows Operating systems, Linux Kernel 2.2.x & 2.4.x, Linux Kernel 2.0.x, and the other *NIX based group.

Operating System	IP TTL value with Echo requests	IP TTL value with Echo replies
Microsoft Windows Family	32	128
Other *NIX based	255	255
LINUX Kernel 2.2.x & 2.4.x	64	255
LINUX Kernel 2.0.x	64	64
Microsoft Windows 2000	128	128
Microsoft Windows 95	32	32

Table 16: Further dividing the groups of operating systems according to IP TTL field value in the ICMP ECHO Requests and in the ICMP ECHO Replies

One would expect that the IP TTL field value would be the same with both ICMP query requests and ICMP query replies. Apparently this is not true and providing us with valuable information about the operating system querying / being queried.

Using the IP TTL field value as the sole parameter to distinguish between oprating systems is not enough. It can be used as another criteria when using other methods, but not as a sole criterion.

7.1.6 Using Fragmented ICMP Address Mask Requests[49]

Identifying Sun Solaris & HPUX 11.0x based machines

Only some operating systems will answer an ICMP Address Mask request. These operating systems include – ULTRIX, OpenVMS, Windows 95/98/98 SE/ME, NT below SP 4, HPUX 11.0x and Sun Solaris.

How can we distinguish between those who answer the request?

This is a regular ICMP Address Mask request sent with the `sing` utility to a Sun Solaris 2.7 machine:

```
[root@aik icmp]# ./sing -mask IP_Address
SINGing to IP_Address (IP_Address): 12 data bytes
12 bytes from IP_Address: icmp_seq=0 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=1 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=2 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=3 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=4 ttl=236 mask=255.255.255.0

--- IP_Address sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
```

[49] The Solaris portion was also discovered by Alfredo Andres Omella.

All operating systems that will answer with ICMP Address Mask reply will reply with the Address Mask of the network they reside on.

What will happen if we will introduce a little twist? Lets say we would send those queries fragmented?

In the next example, I have sent an ICMP Address Mask request to the same Sun Solaris 2.7 host, this time fragmented to pieces of 8 bytes of IP data. As we can see the answer I got was unusual:

```
[root@aik icmp]# ./sing -mask -c 2 -F 8 IP_Address
SINGing to IP_Address (IP_Address): 12 data bytes
12 bytes from IP_Address: icmp_seq=0 ttl=241 mask=0.0.0.0
12 bytes from IP_Address: icmp_seq=1 ttl=241 mask=0.0.0.0

--- IP_Address sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
```

The `tcpdump` trace:

```
20:02:48.441174 ppp0 > y.y.y.y > Host_Address: icmp: address mask
request (frag 13170:8@0+)
                         4500 001c 3372 2000 ff01 50ab yyyy yyyy
                         xxxx xxxx 1100 aee3 401c 0000
20:02:48.442858 ppp0 > y.y.y.y > Host_Address: (frag 13170:4@8)
                         4500 0018 3372 0001 ff01 70ae yyyy yyyy
                         xxxx xxxx 0000 0000
20:02:49.111427 ppp0 < Host_Address > y.y.y.y: icmp: address mask is
0x00000000 (DF)
                         4500 0020 3618 4000 f101 3c01 xxxx xxxx
                         yyyy yyyy 1200 ade3 401c 0000 0000 0000
```

The same Sun Solaris 2.7 based host now replies with 0.0.0.0 as the Address Mask for the network it resides on. The same behavioral patterns were produced against an HPUX 11.0x operating system based machine[50]. In the next example I have tested this behavior against an HPUX B.11.0 based machine:

```
[root@godfather /root]# sing -mask -c 2 172.18.1.5
SINGing to 172.18.1.5 (172.18.1.5): 12 data bytes
12 bytes from 172.18.1.5: seq=0 DF! ttl=254 TOS=0 mask=255.255.255.0
12 bytes from 172.18.1.5: seq=1 DF! ttl=254 TOS=0 mask=255.255.255.0

--- 172.18.1.5 sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
```

[50] When I have published this information in Bugtraq (August 5, 2000) Peter J. Holzer notified me that HP-UX 11.00 produce the same behavior as the SUN Solaris boxes. Darren Reed also noted that because Sun Solaris and HP-UX 11.0 share the same third party (Mentat) implementation for some of their TCP/IP stacks this behavior is produced by both.

```
[root@localhost /root]# sing -mask -c 2 -F 8 172.18.1.5
SINGing to 172.18.1.5 (172.18.1.5): 12 data bytes
12 bytes from 172.18.1.5: seq=0 DF! ttl=254 TOS=0 mask=0.0.0.0
12 bytes from 172.18.1.5: seq=1 DF! ttl=254 TOS=0 mask=0.0.0.0

 --- 172.18.1.5 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
```

The `tcpdump` trace:

```
17:49:41.947465  eth0 > 172.18.2.201 > 172.18.1.5: icmp: address mask
request (frag 13170:8@0+) (ttl 255)
                         4500 001c 3372 2000 ff01 0c7c ac12 02c9
                         ac12 0105 1100 d3f5 1b0a 0000
17:49:41.957465 eth0 > 172.18.2.201 > 172.18.1.5: (frag 13170:4@8) (ttl
255)
                         4500 0018 3372 0001 ff01 2c7f ac12 02c9
                         ac12 0105 0000 0000
17:49:41.957465 eth0 < 172.18.1.5 > 172.18.2.201: icmp: address mask is
0x00000000 (DF) (ttl 254, id 7188)
                         4500 0020 1c14 4000 fe01 04d6 ac12 0105
                         ac12 02c9 1200 d2f5 1b0a 0000 0000 00
```

What will happen with the other operating systems, how will they reply?

They all will respond with the real Address Mask in their replies.

Here we got a distinction between SUN Solaris & HP-UX 11.x based machines to the other operating systems that will answer for ICMP Address Mask request.

Important notice: When I have tested this method I have encountered some problems replicating the results with different ISPs. As it seems from analyzing the information I got, certain ISPs would block fragmented ICMP datagrams. This behavior would not enable this method to succeed. One way of testing this is to send a regular ICMP Echo request. We should watch for a response from the probed machine. If received, than we should send ICMP Echo request, this time fragmented. If no reply is received than your ISP is blocking ICMP fragments probably.

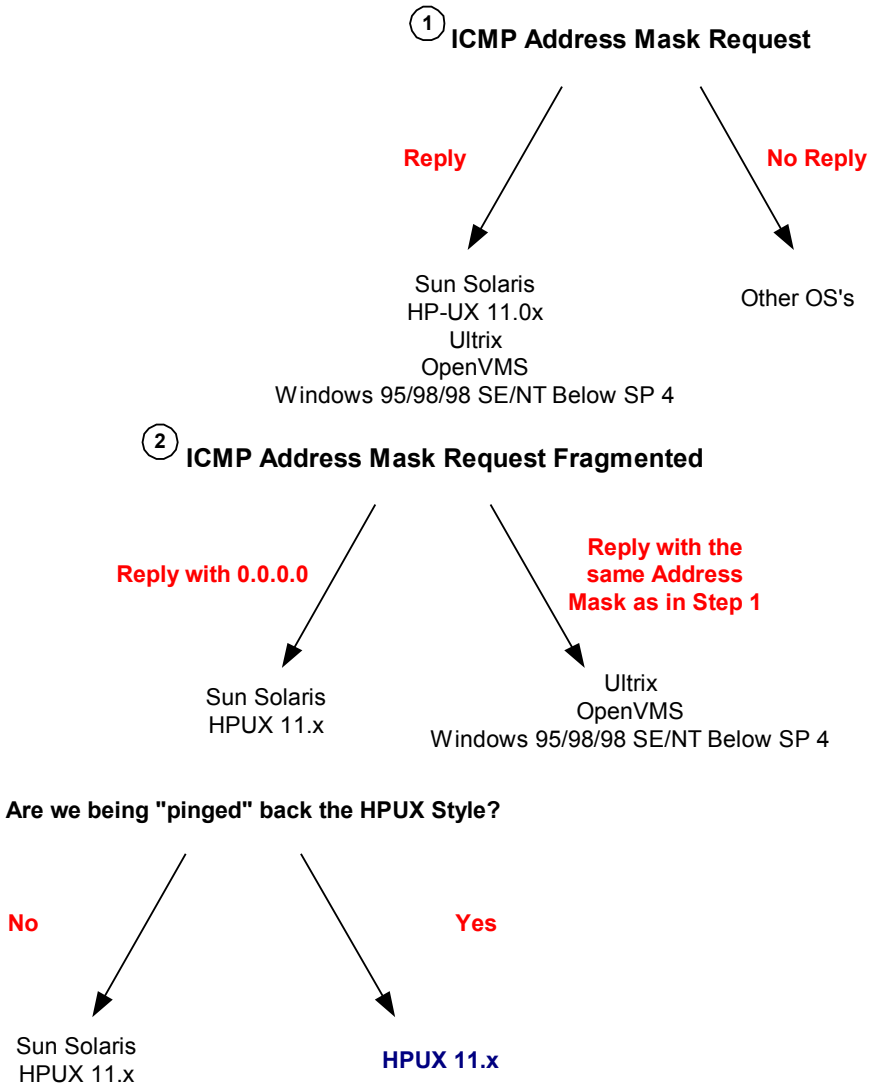


Diagram 2: Finger Printing Using ICMP Address Mask Requests

7.2 Using Crafted ICMP Query Messages

Playing with the TOS Field

Each IP Datagram has an 8-bit field called the “TOS Byte”, which represents the IP support for prioritization and Type-of-Service handling.

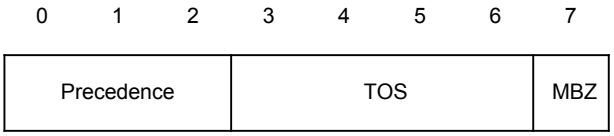


Figure 26: The Type of Service Byte

The “TOS Byte” consists of three fields.

The “Precedence field”, which is 3-bit long, is intended to prioritize the IP Datagram. It has eight levels of prioritization[51]:

Precedence	Definition
0	Routine (Normal)
1	Priority
2	Immediate
3	Flash
4	Flash Override
5	Critical
6	Internetwork Control
7	Network control

Table 17: Precedence Field Values

Higher priority traffic should be sent before lower priority traffic.

The second field, 4 bits long, is the “Type-of-Service” field. It is intended to describe how the network should make tradeoffs between throughput, delay, reliability, and cost in routing an IP Datagram.

RFC 1349[52] has defined the “Type-of-Service” field as a single enumerated value, thus interpreted as a numeric value rather than independent flags (with RFC 791 the 4 bits were distinct options, allowing combinations as well). The 4 bits represents a maximum of 16 possible values.

Value (Hex)	Value (Dec)	Value (Binary)	Service
0	0	0000	Normal
1	1	1000	Minimize Delay
2	2	0100	Maximize Throughput
4	4	0010	Maximize Reliability
8	8	0001	Minimize Cost
F	15	1111	Maximize Security[53]

Table 18: Type-of-Service Field Values

What about the other 10 value possibilities?

RFC 1349 refer to this issue and states that “although the semantics of values other than the five listed above are not defined by this memo, they are perfectly legal TOS values, and hosts and routers must not preclude their use in any way”…”A host or a router need not make any

[51] RFC 791 – Internet Protocol, http://www.ietf.org/rfc/rfc791.txt.
[52] RFC 1349 - Type of Service in the Internet Protocol Suite, http://www.ietf.org/rfc/rfc1349.txt.
[53] RFC 1455 - Physical Link Security Type of Service, http://www.ietf.org/rfc/rfc1455.txt.

distinction between TOS values who's semantics are defined by this memo and those that are not".

The last field, the "MBZ" (must be zero), is unused and must be zero. Routers and hosts ignore this last field. This field is 1 bit long.

Combining Type-of-Service flags with the different prioritization values, dictates very explicit types of behavior with certain types of data.

Please note that not all TCP/IP implementations will use these values (nor offer a mechanism for setting these values), and some will not handle datagrams which have Type-of-Service and/or Precedence values other than the defaults, differently.

7.2.1 Precedence Bits Echoing

Identifying Microsoft Windows 2000, ULTRIX, HPUX 11.0&10.30, OpenVMS

The precedence bits behavior is a problem. RFC 1122, which defines the requirements for Internet Hosts, does not outline the way to handle the Precedence Bits with ICMP. The RFC only statement about the Precedence Bits is:

> "The Precedence field is intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of this document and the IP standard specification. Vendors should consult the Defense Communication Agency (DCA) for guidance on the IP Precedence field and its implications for other protocol layers. However, vendors should note that the use of precedence will most likely require that its value be passed between protocol layers in just the same way as the TOS field is passed".

This does not give us something to work with.

RFC 1812, Requirements for IP version 4 routers state that:

> "An ICMP reply message MUST have its IP Precedence field set to the value as the IP Precedence field in the ICMP request that provoked the reply".

Echoing back the Precedence field value has its logic, because the TOS field should be echoed back with an ICMP query replies, and both the Precedence field and the TOS field were to dictate very explicit types of behavior with certain types of data.

As you can understand we do not have a clear ruling about this issue. I was thinking it might be a ground for an operating system fingerprinting method.

Most of the operating systems I have checked will behave as the next behavioral example with AIX 4.3. With the next example an ICMP Echo request is sent with one of the precedence bits set:

```
[root@godfather /root]# /usr/local/bin/sing -c 5 -TOS 128 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=239 TOS=128 time=5896.472 ms
16 bytes from y.y.y.y: seq=1 ttl=239 TOS=128 time=5952.071 ms
16 bytes from y.y.y.y: seq=2 ttl=239 TOS=128 time=6102.020 ms
16 bytes from y.y.y.y: seq=3 ttl=239 TOS=128 time=6261.997 ms
```

```
16 bytes from y.y.y.y: seq=4 ttl=239 TOS=128 time=5842.726 ms

--- y.y.y.y sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 5842.726/6011.057/6261.997 ms
```

The `tcpdump` trace:

```
21:02:53.241666 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                  4580 0024 3372 0000 ff01 619c xxxx xxxx
                  yyyy yyyy 0800 c278 6f05 0000 dd97 0d3a
                  d8af 0300

21:02:59.134297 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x80]
(ttl 239, id 40656)
                  4580 0024 9ed0 0000 ef01 063e yyyy yyyy
                  xxxx xxxx 0000 ca78 6f05 0000 dd97 0d3a
                  d8af 0300
```

The AIX 4.3 based machine queried is using the precedence bits value used for the ICMP Echo request with its ICMP Echo reply.

Some operating systems are the exception.

The next example is with Microsoft Windows 2000. The same ICMP Echo Request was sent:

```
[root@godfather /]# /usr/local/bin/sing -c 5 -TOS 128 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=111 TOS=0 time=6261.043 ms
16 bytes from y.y.y.y: seq=1 ttl=111 TOS=0 time=6422.019 ms
16 bytes from y.y.y.y: seq=2 ttl=111 TOS=0 time=6572.675 ms
16 bytes from y.y.y.y: seq=4 ttl=111 TOS=0 time=6282.022 ms

--- y.y.y.y sing statistics ---
5 packets transmitted, 4 packets received, 20% packet loss
round-trip min/avg/max = 6261.043/6384.440/6572.675 ms
```

The `tcpdump` trace:

```
20:13:36.717070 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                  4580 0024 3372 0000 ff01 d95d xxxx xxxx
                  yyyy yyyy 0800 df43 c304 0000 508c 0d3a
                  edf0 0a00

20:13:42.974295 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 111, id
26133)
                  4500 0024 6615 0000 6f01 373b yyyy yyyy
                  xxxx xxxx 0000 e743 c304 0000 508c 0d3a
                  edf0 0a00
```

The ICMP Echo reply will not use the value assigned to the Precedence Bits with the ICMP Echo Request.

Which operating systems share this behavioral pattern? Microsoft Windows 2000 Family, and ULTRIX.

Differentiating between Microsoft Windows 2000 and Ultrix is easily achieved if we examine the IP TTL field values. With ULTRIX the value assigned to the ICMP Echo reply will be 255, with Microsoft Windows 2000 it will be 128.

Another interesting case is with HPUX 11.0. Lets examine the trace and logs:

```
[root@godfather /]# /usr/local/bin/sing -c 2 -TOS 128 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=242 TOS=128 time=639.274 ms
16 bytes from y.y.y.y: seq=1 DF! ttl=242 TOS=0 time=310.427 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 310.427/474.850/639.274 ms
```

The first reply from the HPUX machine echoed back the precedence field value we were using with the ICMP Echo Request. But what have happened between the first and the second reply?

```
00:35:09.315260 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                  4580 0024 3372 0000 ff01 4bd1 xxxx xxxx
                  yyyy yyyy 0800 16f0 db3c 0000 9dc9 0d3a
                  56cf 0400

00:35:09.944274 ppp0 < y.y.y.y > x.x.x.x: icmp: echo request (DF) (ttl
242, id 22417)
                  4500 05dc 5791 4000 f201 ef79 yyyy yyyy
                  xxxx xxxx 0800 7e52 9abc def0 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
                  0000 0000 0000 0000 0000 0000 0000 0000
```

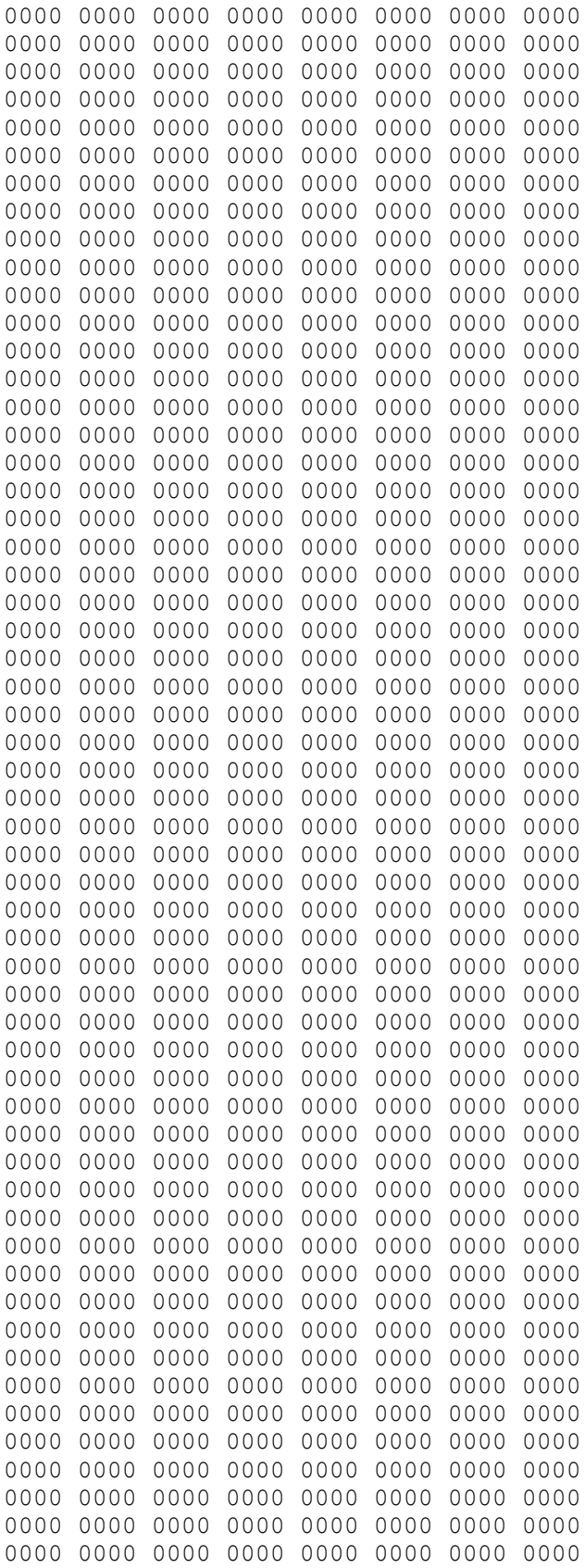
```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000
```

The first request was sent, as an instant reply the HPUX 11.0 machine initiated its PMTU discovery process with ICMP Echo requests and sent an ICMP Echo request 1500 bytes long[54].

```
00:35:09.944355 ppp0 > x.x.x.x > y.y.y.y: icmp: echo reply (ttl 255, id
14194)
                         4500 05dc 3772 0000 ff01 4299 xxxx xxxx
                         yyyy yyyy 0000 8652 9abc def0 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

[54] For an explanation about the HPUX 11.0 PMTU process using ICMP Echo Requests please see the "DF Playground" section.



```
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000

00:35:09.954282 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x80]
(ttl 242, id 22418)
                    4580 0024 5792 0000 f201 34b1 yyyy yyyy
                    xxxx xxxx 0000 1ef0 db3c 0000 9dc9 0d3a
                    56cf 0400
```

The ICMP Echo reply received from the HPUX 11.0 machine for the ICMP Echo request echoed back the Precedence bits field value.

Another ICMP Echo request was sent with TOS byte field value of 0x80 hex:

```
00:35:10.314321 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                    4580 0024 3372 0000 ff01 4bd1 xxxx xxxx
                    yyyy yyyy 0800 b7f3 db3c 0100 9ec9 0d3a
                    b3cb 0400

00:35:10.624275 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (DF) (ttl
242, id 22419)
                    4500 0024 5793 4000 f201 f52f yyyy yyyy
                    xxxx xxxx 0000 bff3 db3c 0100 9ec9 0d3a
                    b3cb 0400
```

This time the ICMP Echo reply received did not echo back the TOS byte field value. The DF bit was set. The PMTU discovery process finished its initial stages and went to regular operation. From now on the ICMP Echo replies did not echo the Precedence bits field value.

This gives us the ability to track down HPUX 11.0 (and 10.30) based machines when they are using the PMTU Discovery process.

This is not the only behavioral pattern I have experienced with HPUX 11.x based machines:

```
[root@godfather /]# sing -echo -c 2 -TOS 128 172.18.1.5
SINGing to 172.18.1.5 (172.18.1.5): 16 data bytes
16 bytes from 172.18.1.5: seq=0 DF! ttl=254 TOS=128 time=4.659 ms
16 bytes from 172.18.1.5: seq=1 DF! ttl=254 TOS=128 time=4.160 ms

--- 172.18.1.5 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
```

```
round-trip min/avg/max = 4.160/4.410/4.659 ms
```

The `tcpdump` trace:

```
17:52:02.197465 eth0 > 172.18.2.201 > 172.18.1.5: icmp: echo request
[tos 0x80]  (ttl 255, id 13170)
                         4580 0024 3372 0000 ff01 2bf4 ac12 02c9
                         ac12 0105 0800 24bf 1e0a 0000 12da 073b
                         9821 0300
17:52:02.197465 eth0 < 172.18.1.5 > 172.18.2.201: icmp: echo reply (DF)
[tos 0x80]  (ttl 254, id 7190)
                         4580 0024 1c16 4000 fe01 0450 ac12 0105
                         ac12 02c9 0000 2cbf 1e0a 0000 12da 073b
                         9821 0300
```

Therefore it is important to identify where the PMTU discovery process using ICMP Echo requests is being used and when it is not. We may experience different results with an HPUX 11.x based machine.

7.2.1.1 Changed Pattern with other ICMP Query Message Types

We can identify change of pattern with OpenVMS, Microsft Windows 98, 98SE, and ME. With ICMP Echo replies they all would echo back the Precedence bits value, but with ICMP Timestamp replies they will change the behavior and send back 0x000.

Since OpenVMS use 255 as its IP TTL field value, and the Microsoft Windows based machines use 128, we can differentiate between them and isolate OpenVMS, and the Microsoft based OSs.

Further distinction between the Microsoft operating systems can be achieved if we will query them with ICMP Address Mask request where only Microsoft Windows 98/98SE will answer for. The Microsoft Windows ME will not reply, enabling us to identify it.

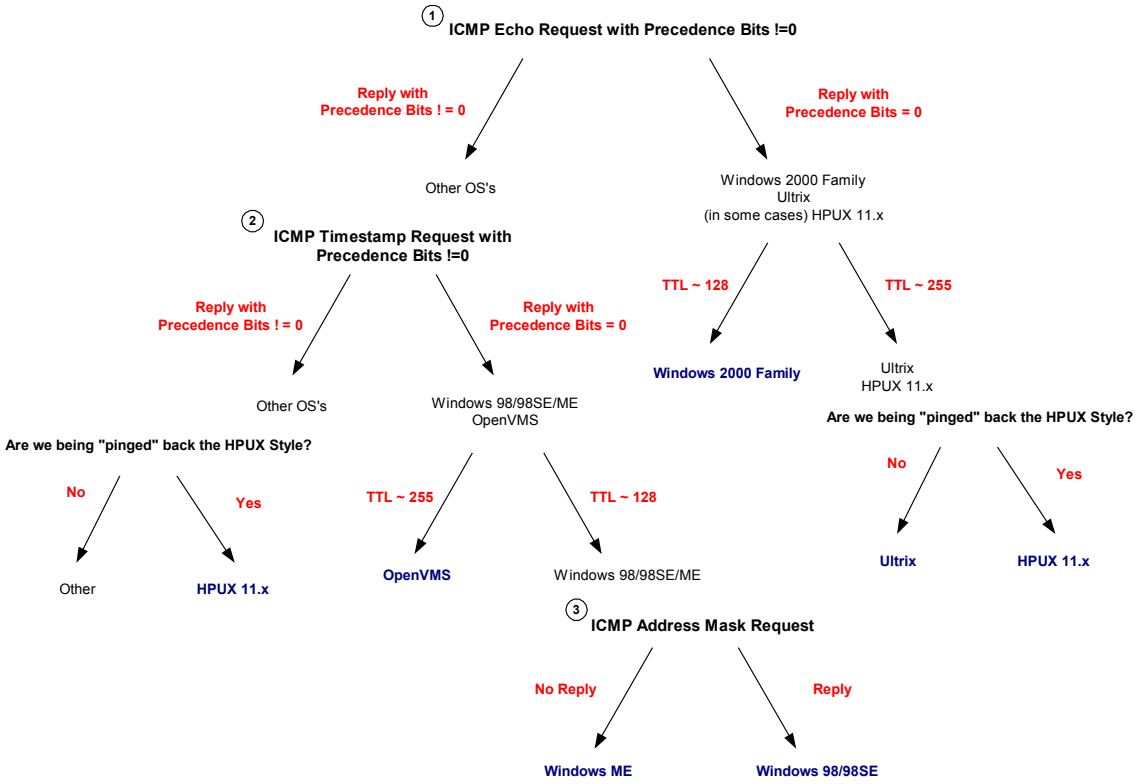


Diagram 3: An example for a way to fingerprint Microsoft Windows 2000, Ultrix, HPUX 11.0 & 10.30, OpenVMS, Microsoft Windows ME, and Microsoft Windows 98/98SE based machines with ICMP Query messages with the Precedence Bits field !=0

An HPUX based machine is placed in both sides of the diagram. If the PMTU Discovery process will be faster than the first answering ICMP Echo reply than we might have an ICMP Echo reply with Precedence bits equal to zero answering an ICMP Echo request with a value different than zero as its precedence bits value. On the other hand, we demonstrated cases in which an HPUX based machines will echo back any value the Precedence bits will carry with an ICMP Echo request.

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
Linux Kernel 2.4.x	Not Answering	!=0x00	Not Answering	!=0x00
Linux Kernel 2.2.x	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.1.1	Not Answering	!=0x00	Not Answering	
OpenBSD 2.7	Not Answering		Not Answering	!=0x00
OpenBSD 2.6	Not Answering		Not Answering	!=0x00
NetBSD	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering		Not Answering	!=0x00

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00 -> 0x00	!=0x00 -> 0x00
Compaq Tru64 v5.0	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.3	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.2.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 3.2	!=0x00	!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5	0x00	0x00	0x00	0x00
OpenVMS v7.1-2	0x00	0x00	0x00	!=0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x00	0x00	!=0x00
Windows 98 SE	Not Answering	0x00	0x00	!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Table 19: ICMP Query Message Types with Precedence Bits ! = 0

7.2.2 TOSing OSs out of the Window / "TOS Echoing"
Identifying Microsoft Windows 2000, Ultrix, and Novell Netware

7.2.2.1 The use of the Type-of-Service field with the ICMP Protocol
RFC 1349 also defines the usage of the Type-of-Service field with the ICMP messages. It distinguishes between ICMP error messages (Destination Unreachable, Source Quench, Redirect, Time Exceeded, and Parameter Problem), ICMP query messages (Echo, Router Solicitation, Timestamp, Information request, Address Mask request) and ICMP reply messages (Echo reply, Router Advertisement, Timestamp reply, Information reply, Address Mask reply).

Simple rules are defined:

- An ICMP error message is always sent with the default TOS (0x0000)
- An ICMP request message *may* be sent with any value in the TOS field. "A mechanism to allow the user to specify the TOS value to be used would be a useful feature in many applications that generate ICMP request messages"[55].

[55] RFC 1349 - Type of Service in the Internet Protocol Suite, http://www.ietf.org/rfc/rfc1349.txt.

The RFC further specify that although ICMP request messages are normally sent with the default TOS, there are sometimes good reasons why they would be sent with some other TOS value.

- An ICMP reply message is sent with the same value in the TOS field as was used in the corresponding ICMP request message.

Using this logic I have decided to check if certain operating systems react correctly to an ICMP query messages with a Type-of-Service field value, which is different than the default (0x0000).

The check out was produced with all ICMP query message types, sent with a Type-of-Service field set to a known value, then set to an unknown value (the term known and unknown are used here because I was not experimenting with non-legit values, and since any value may be sent inside this field).

The following example is an ICMP Echo request sent to my FreeBSD 4.0 machine with the TOS field equals an 8 hex value, which is a legit TOS value. The utility used here is `sing`[56]:

```
[root@godfather /]# ./sing -echo -TOS 8 IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=8 time=260.043 ms
16 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=8 time=180.011 ms
16 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=8 time=240.240 ms
16 bytes from IP_Address: icmp_seq=5 ttl=243 TOS=8 time=260.037 ms
16 bytes from IP_Address: icmp_seq=6 ttl=243 TOS=8 time=290.033 ms

--- IP_Address sing statistics ---
7 packets transmitted, 5 packets received, 28% packet loss
round-trip min/avg/max = 180.011/246.073/290.033 ms
[root@godfather /]#
```

The `tcpdump` trace:

```
17:23:46.605297 if 4  > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x8]
(ttl 255, id 13170)
                  4508 0024 3372 0000 ff01 60e4 xxxx xxxx
                  yyyy yyyy 0800 0e9a d604 0600 f2ea bc39
                  553c 0900
17:23:46.895255 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x8]
(ttl 243, id 58832)
                  4508 0024 e5d0 0000 f301 ba85 yyyy yyyy
                  xxxx xxxx 0000 169a d604 0600 f2ea bc39
                  553c 0900
```

This is the second test I have produced, sending ICMP Echo request with the Type-of-Service field set to a 10 Hex value, a value that is not a known Type-of-Service value:

[56] sing has the ability to monitor for any replies and than print the received TOS value. I find this option very useful, and thank the author for embedding this function, as I requested.

```
[root@godfather bin]# ./sing -echo -TOS 10 IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=10 time=197.933 ms
16 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=10 time=340.048 ms
16 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=10 time=250.025 ms
...

--- IP_Address sing statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 197.933/259.726/340.048 ms
```

The `tcpdump` trace:

```
17:24:36.155298 if 4  > y.y.y.y > x.x.x.x: icmp: echo request [tos
0xa,ECT]  (ttl 255, id 13170)
                  450a 0024 3372 0000 ff01 60e2 yyyy yyyy
                  xxxx xxxx 0800 af77 d904 0600 24eb bc39
                  865e 0200
17:24:36.415254 if 4  < x.x.x.x > y.y.y.y: icmp: echo reply [tos
0xa,ECT]  (ttl 243, id 65031)
                  450a 0024 fe07 0000 f301 a24c xxxx xxxx
                  yyyy yyyy 0000 b777 d904 0600 24eb bc39
                  865e 0200
```

As it can be seen from the tcpdump trace, the ICMP echo reply message was sent with the same value in the TOS field as was used in the corresponding ICMP echo request message.

I had to verify this behavioral pattern with FreeBSD 4.0 with the other ICMP query messages it answers for. Since FreeBSD 4.0 does not respond to ICMP Information requests or to ICMP Address Mask requests I had to verify this with ICMP Timestamp requests only.

Again the utility I was used is `sing`:

```
[root@godfather /]# ./sing -tstamp -TOS 8 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=8 diff=6832668
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=8 diff=6832403
20 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=8 diff=6832633
20 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=8 diff=6832605
20 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=8 diff=6832431

--- IP_Address sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
[root@godfather /]#
```

The `tcpdump` trace:

```
17:26:00.455295 if 4  > x.x.x.x > y.y.y.y: icmp: time stamp request
[tos 0x8]  (ttl 255, id 13170)
```

```
                    4508 0028 3372 0000 ff01 60e0 xxxx xxxx
                    yyyy yyyy 0d00 345b dd04 0400 0318 da87
                    0000 0000 0000 0000
17:26:00.755254 if 4  < y.y.y.y > x.x.x.x: icmp: time stamp reply [tos
0x8]  (ttl 243, id 5867)
                    4508 0028 16eb 0000 f301 8967 yyyy yyyy
                    xxxx xxxx 0e00 f4ec dd04 0400 0318 da87
                    0380 1bb7 0380 1bb7
```

The second test with was performed with the TOS field value set to 10 Hex value:

```
[root@godfather /]# ./sing -tstamp -TOS 10 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=10 diff=6766872
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=10 diff=6767059
20 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=10 diff=6767059
...

--- IP_Address sing statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
[root@godfather /]#
```

The `tcpdump` trace:

```
17:25:42.548597 if 4  > x.x.x.x > y.y.y.y: icmp: time stamp request
[tos 0xa,ECT]  (ttl 255, id 13170)
                    450a 0028 3372 0000 ff01 60de xxxx xxxx
                    yyyy yyyy 0d00 7f4e dc04 0000 0318 9494
                    0000 0000 0000 0000
17:25:42.795254 if 4  < y.y.y.y > x.x.x.x: icmp: time stamp reply [tos
0xa,ECT]  (ttl 243, id 3519)
                    450a 0028 0dbf 0000 f301 9291 yyyy yyyy
                    xxxx xxxx 0e00 cbf6 dc04 0000 0318 9494
                    037f d5ac 037f d5ac
```

The same behavior was produced. The ICMP Timestamp replies were sent with the TOS field value equals the TOS field value of the ICMP Timestamp requests.

Ok. I was curious again. I imagined that the Microsoft Windows implementation of the things might be a little different.

When I was examining ICMP Echo requests I noticed something is wrong with the Microsoft implementation:

```
[root@godfather /root]# sing -echo -c 2 -TOS 8 172.18.2.200
SINGing to 172.18.2.200 (172.18.2.200): 16 data bytes
16 bytes from 172.18.2.200: seq=0 ttl=128 TOS=0 time=4.535 ms
16 bytes from 172.18.2.200: seq=1 ttl=128 TOS=0 time=4.088 ms
```

```
--- 172.18.2.200 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 4.088/4.311/4.535 ms
[root@godfather /root]#
```

The `snort` trace:

```
05/20/01-18:04:32.507465  172.18.2.201 -> 172.18.2.200
ICMP TTL:255 TOS:0x8 ID:13170 IpLen:20 DgmLen:36
Type:8  Code:0  ID:11274   Seq:0  ECHO
00 DD 07 3B A0 E0 07 00                          ...;....

05/20/01-18:04:32.507465  172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:44503 IpLen:20 DgmLen:36
Type:0  Code:0  ID:11274   Seq:0  ECHO REPLY
00 DD 07 3B A0 E0 07 00                          ...;....
```

Oops! Some one zero out my Type-of-Service field value!

This example was produced against a Microsoft Windows 2000 Professional SP2 based machine. All the Microsoft Windows 2000 family based machines (Professional, Server, Advanced Server) will act the same.

The other Microsoft based operating systems will act correctly - Microsoft Windows 98/SE/ME, Microsoft Windows NT 4 Workstation SP3, Microsoft Windows NT 4 Server SP4, Microsoft Windows NT 4 Workstation SP6a.

Ultrix and Novell Netware will share the same behavioral pattern as with Microsoft Windows 2000 family.

How can we distinguish between those?
If we will look at the IP TTL field values carried with the ICMP echo replies we can divide the group into two. The original value an Ultrix ICMP Echo reply datagram will have for its IP TTL field value will be 225, while Microsoft Windows 2000 and Novell Netware will use 128.

The next step will be to query the questionable IP Addresses of Microsoft Windows 2000 and Novell Netware with ICMP Timestamp request. The Microsoft Windows 2000 based machines will answer the query while the Novell Netware based machines will not.

Other methods to distinguish between the Microsoft Windows 2000 based machines to Novell Netware based machine may apply here as well.

7.2.2.2 Changed Pattern with Other ICMP Message Types

Not all Microsoft based operating systems will maintain a single behavioral pattern with all ICMP query requests. Some of the Microsoft based operating systems will change their behavior experienced with ICMP echo replies, and with ICMP timestamp replies they will zero out the TOS field value although in the ICMP timestamp requests a value different then zero was received.

The named operating systems are Microsoft Windows 98/98SE/ME.

Microsoft Windows 2000 based machines will maintain the same behavioral pattern regarding the TOS field with the ICMP Timestamp mechanism.

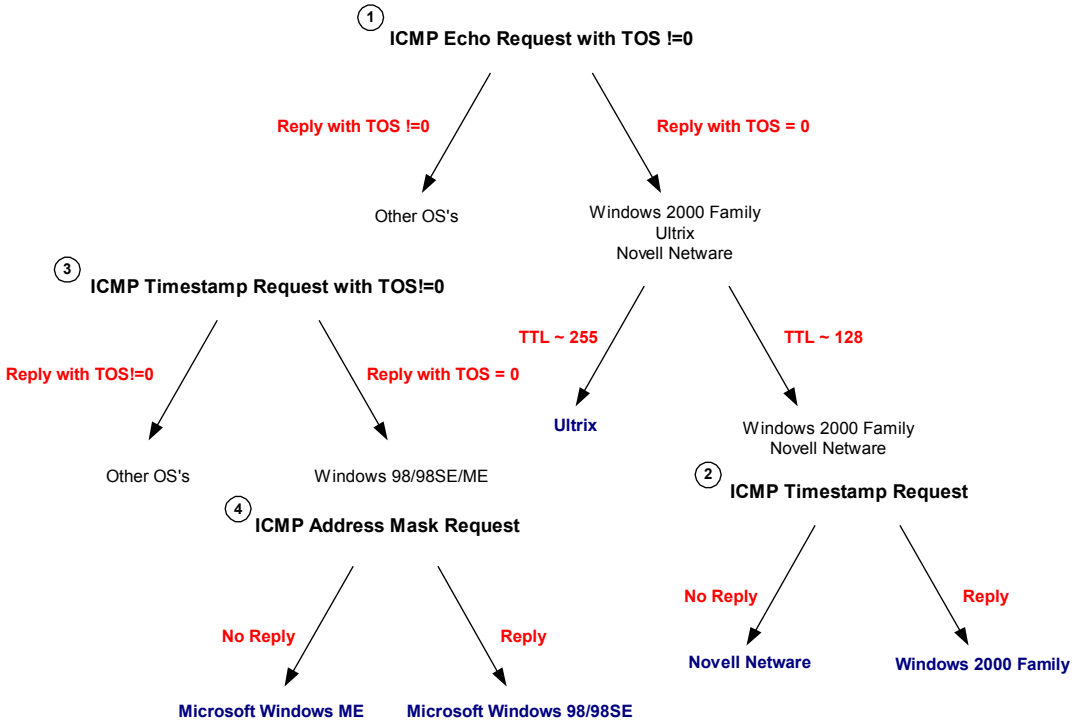


Diagram 4: An example for a way to fingerprint Microsoft Windows 2000, Ultrix, Novell Netware, Microsoft Windows ME, and Microsoft Windows 98/98SE based machines with ICMP query messages with the TOS bits field !=0

Operating System	Information Request With TOS!=0x00	Time Stamp Request With TOS!=0x00	Address Mask Request With TOS!=0x00	Echo Request With TOS!=0x00
Linux Kernel 2.4.x	Not Answering	!=0x00	Not Answering	!=0x00
Linux Kernel 2.2.x	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 3.4	Not Answering		Not Answering	
OpenBSD 2.7	Not Answering	!=0x00	Not Answering	!=0x00
OpenBSD 2.6	Not Answering	!=0x00	Not Answering	!=0x00
NetBSD	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering	!=0x00	Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented			
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00

Operating System	Information Request With TOS!=0x00	Time Stamp Request With TOS!=0x00	Request With TOS!=0x00	Echo Request
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00	!=0x00
Compaq Tru64 v5.0		!=0x00	Not Answering	!=0x00
Irix 6.5.3	Not Answering	!=0x00	Not Answering	!=0x00
Irix 6.5.8	Not Answering	!=0x00	Not Answering	!=0x00
AIX 4.1		!=0x00	Not Answering	!=0x00
AIX 3.2		!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5		0x00	0x00	0x00
OpenVMS v7.1-2		!=0x00	!=0x00	!=0x00
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x00	0x00	!=0x00
Windows 98 SE	Not Answering	0x00		!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Table 20: ICMP Query Message Types with TOS! = 0

7.2.3 Using the TOS Byte's Unused Bit

Identifying Microsoft Windows 2000, ULTRIX and Others

RFC 1349 states that the last field of the TOS byte, the "MBZ" (must be zero), is unused and must be zero. The RFC also states that routers and hosts ignore the value of this bit.

This is the only statement about the unused bit in the TOS Byte in the RFCs. The RFC states: "The originator of a datagram sets this field to Zero".

Obviously it was meant that this field would be always zero. But what will happen if we would set this bit with our ICMP Echo requests? Will this bit be zero out on reply or will it be echoed back?

Only with ICMP Echo requests we can have a clear identification of OSs.

The next example is an ICMP Echo request sent with the Unused bit in the TOS Byte set, targeting a FreeBSD 4.1.1 machine:

```
[root@godfather /root]# /usr/local/bin/sing  -c 2 -TOS 1 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=233 TOS=1 time=330.461 ms
16 bytes from y.y.y.y: seq=1 ttl=233 TOS=1 time=723.300 ms

--- y.y.y.y sing statistics ---
```

```
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 330.461/526.880/723.300 ms
[root@godfather /root]#
```

Echoing back the Unused bit in the TOS Byte represents the behavior of most of the operating systems I have checked this behavior against.

Which operating systems are the exceptions, and will not echo back the Unused bit in the TOS byte if set?

The next example is with Microsoft Windows 2000 Professional SP2 as the targeted machine:

```
[root@godfather /root]# sing -echo -c 2 -TOS 1 172.18.2.200
SINGing to 172.18.2.200 (172.18.2.200): 16 data bytes
16 bytes from 172.18.2.200: seq=0 ttl=128 TOS=0 time=4.519 ms
16 bytes from 172.18.2.200: seq=1 ttl=128 TOS=0 time=4.101 ms

--- 172.18.2.200 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 4.101/4.310/4.519 ms
[root@godfather /root]#
```

The `snort` trace:

```
05/20/01-18:14:18.707465  172.18.2.201 -> 172.18.2.200
ICMP TTL:255 TOS:0x1 ID:13170 IpLen:20 DgmLen:36
Type:8  Code:0  ID:13578   Seq:0  ECHO
4A DF 07 3B FC D8 0A 00                         J..;....


05/20/01-18:14:18.707465  172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:45260 IpLen:20 DgmLen:36
Type:0  Code:0  ID:13578  Seq:0  ECHO REPLY
4A DF 07 3B FC D8 0A 00                         J..;....
```

Another OS that behaves the same is Ultrix:

```
[root@godfather /]# /usr/local/bin/sing -c 2 -TOS 1 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=237 TOS=0 time=371.776 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 1 packets received, 50% packet loss
round-trip min/avg/max = 371.776/371.776/371.776 ms
[root@godfather /]#
```

We will use, again, the IP TTL field value to differentiate between Microsoft Windows 2000 (128) and Ultrix (255).

7.2.3.1 Changed Pattern with Replies for Different ICMP Query Types

We have a changed pattern with Microsoft Windows 98/98SE/ME when using other ICMP query message types other than ICMP Echo requests. Instead of echoing this field back, this time they will zero out this field value with their replies.

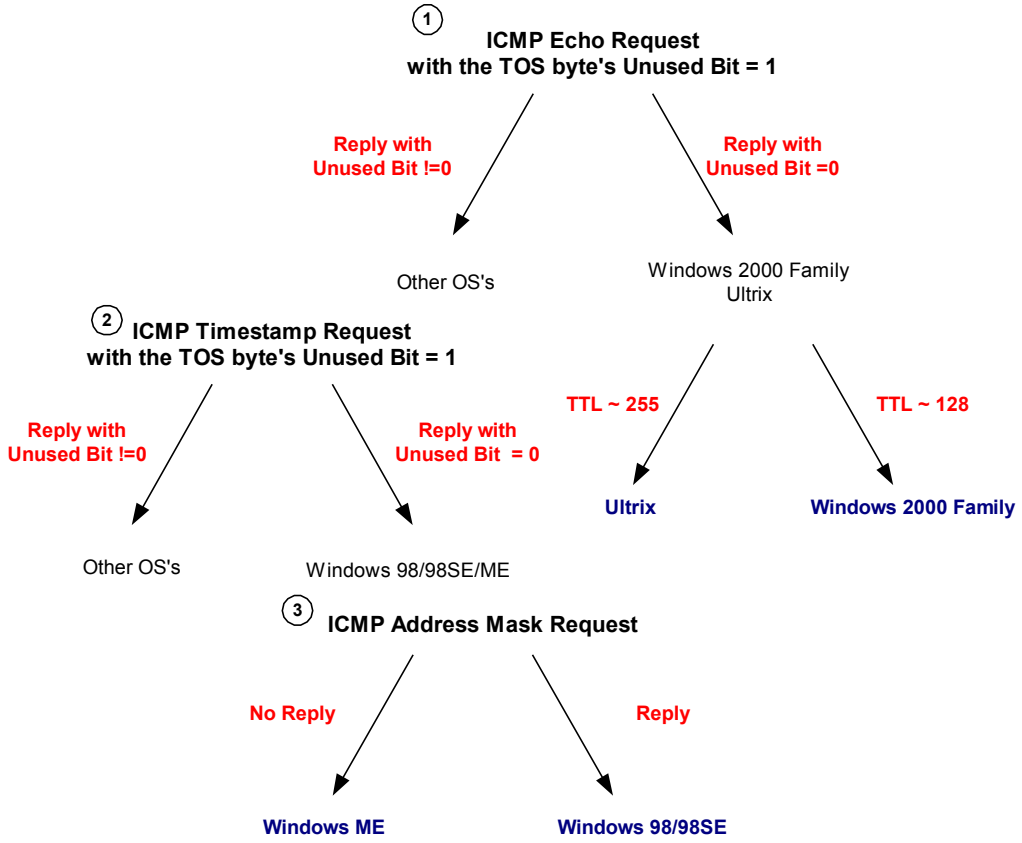


Diagram 5: An example for a way to fingerprint operating systems using the unused bit in the TOS Byte echoing method

Operating System	Information Request With Unused=1	Time Stamp Request With Unused=1	Address Mask Request With Unused=1	Echo Request With Unused=1
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	0x1	Not Answering	0x1
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.0	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.1.1	Not Answering	0x1	Not Answering	0x1
OpenBSD 2.7	Not Answering		Not Answering	

Operating System	Information Request With Unused=1	Time Stamp Request With Unused=1	Address Mask Request With Unused=1	Echo Request With Unused=1
OpenBSD 2.6	Not Answering		Not Answering	
NetBSD	Not Answering		Not Answering	
BSDI BSD/OS 4.0	Not Answering		Not Answering	
BSDI BSD/OS 3.1	Not Answering		Not Answering	
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	0x1	0x1	0x1
Solaris 2.7	Not Implemented	0x1	0x1	0x1
Solaris 2.8	Not Implemented	0x1	0x1	0x1
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	0x1	0x1
Compaq Tru64 v5.0	0x1	0x1	Not Answering	0x1
AIX 4.3	0x1	0x1	Not Answering	0x1
AIX 4.2.1	0x1	0x1	Not Answering	0x1
AIX 4.1	0x1	0x1	Not Answering	0x1
AIX 3.2	0x1	0x1	Not Answering	0x1
ULTRIX 4.2 – 4.5	0x0	0x0	0x0	0x0
OpenVMS v7.1-2	0x1	0x1	0x1	0x1
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x0	0x0	0x1
Windows 98 SE	Not Answering	0x0	0x0	0x1
Windows ME	Not Answering	0x0	Not Answering	0x1
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	0x1
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	
Windows 2000 Professional	Not Answering	0x0	Not Answering	0x0
Windows 2000 Server	Not Answering	0x0	Not Answering	0x0

Table 21: ICMP Query Message Types with the TOS Byte Unused Bit value ! = 0

7.2.4 Using the Unused

Identifying Sun Solaris & HP-UX 10.30 & 11.0x OS based machines

RFC 791 defines a three bits field used for various control flags in the IP Header. Bit 0 of this bits field is the reserved flag, and must be zero according to the RFC.

What will happen if we will decide to break this definition and send our ICMP query requests with this bit set (having the value of one)?

Sun Solaris & HPUX 11.0x (possibly 10.30 as well) will echo back the reserved bit.

In the next example I have sent an ICMP Echo request with the Unused bit set (Reserve Flag), using the –U option of sing, destined an HP-UX B.11.0 based machine:

```
[root@godfather /root]# sing -echo -c 2 -U 172.18.1.5
SINGing to 172.18.1.5 (172.18.1.5): 16 data bytes
16 bytes from 172.18.1.5: seq=0 RF! DF! ttl=254 TOS=0 time=3.037 ms
16 bytes from 172.18.1.5: seq=1 RF! DF! ttl=254 TOS=0 time=2.988 ms
```

```
--- 172.18.1.5 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.988/3.012/3.037 ms
[root@godfather /root]#
```

The `Snort` trace:

```
05/21/01-15:06:05.525407  172.18.2.201 -> 172.18.1.5
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:36 RB
Type:8  Code:0  ID:2822   Seq:0  ECHO
AD 04 09 3B 9D 1C 08 00                         ...;....

05/21/01-15:06:05.525407  172.18.1.5 -> 172.18.2.201
ICMP TTL:254 TOS:0x0 ID:27103 IpLen:20 DgmLen:36 RB DF
Type:0  Code:0  ID:2822  Seq:0  ECHO REPLY
AD 04 09 3B 9D 1C 08 00                         ...;....
```

The following is another behavioral pattern produced against an HPUX 11.0 based machine:

```
21:31:21.033366 if 4  > y.y.y.y > x.x.x.x: icmp: echo request (ttl 255,
id 13170)
                  4500 0024 3372 8000 ff01 fc8c yyyy yyyy
                  xxxx xxxx 0800 8b1b 8603 0000 f924 bd39
                  3082 0000
21:31:21.317916 if 4  < x.x.x.x > y.y.y.y: icmp: echo reply (ttl 236,
id 25606)
                  4500 0024 6406 8000 ec01 def8 xxxx xxxx
                  yyyy yyyy 0000 931b 8603 0000 f924 bd39
                  3082 0000
```

The next example was produced against a Sun Solaris 2.8 based machine:

```
[root@godfather /root]# sing -echo -c 2 -U 172.18.1.12
SINGing to 172.18.1.12 (172.18.1.12): 16 data bytes
16 bytes from 172.18.1.12: seq=0 RF! DF! ttl=254 TOS=0 time=3.716 ms
16 bytes from 172.18.1.12: seq=1 RF! DF! ttl=254 TOS=0 time=2.947 ms

--- 172.18.1.12 sing statistics---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.947/3.332/3.716 ms
[root@godfather /root]#
```

The `tcpdump` trace:

```
15:24:39.975407 eth0 > 172.18.2.201 > 172.18.1.12: icmp: echo request
(ttl 255,id 13170)
```

```
                          4500 0024 3372 8000 ff01 ac6c ac12 02c9
                          ac12 010c 0800 eeaf 1706 0000 0709 093b
                          d305 0f00
15:24:39.985407 eth0 < 172.18.1.12 > 172.18.2.201: icmp: echo reply
(DF) (ttl 254, id 6575)
                          4500 0024 19af c000 fe01 872f ac12 010c
                          ac12 02c9 0000 f6af 1706 0000 0709 093b
                          d305 0f00
```

We might see a distinction between Sun Solaris and HPUX based machines, if the PMTU Discovery process using ICMP Echo requests is enabled on the HPUX queried host. We might see the ICMP Echo reply received, without the DF bit set, and than after we will be queried with an ICMP Echo request 'the HPUX style' back from our target. We might see another case where we will be queried 'the HPUX style' just before the ICMP Echo reply will be received this time with the DF bit set.

We might also see cases in which the ICMP Echo replies from both HPUX and Sun Solaris will be exactly the same.

All ICMP query replies on the same operating system use the same pattern (either echo the reserved bit with all replies or not). This enables us to use another ICMP query message type for this fingerprinting method. If we will send an ICMP Address Mask request with the reserved bit set, the result a Sun Solaris 2.8 machine will produce will be something like this next trace:

```
18:39:32.262869 if 4  > y.y.y.y > x.x.x.x : icmp: address mask request
(ttl 255, id 13170)
                  4500 0020 3372 8000 ff01 e12e yyyy yyyy
                  xxxx xxxx 1100 a0fb 4e04 0000 0000 0000
18:39:32.561373 if 4  < x.x.x.x > y.y.y.y: icmp: address mask is
0xffffff00 (DF) (ttl 243, id 51792)
                  4500 0020 ca50 c000 f301 1650 xxxx xxxx
                  yyyy yyyy 1200 a0fa 4e04 0000 ffff ff00
```

We will have both the reserved bit and the DF bit set with the ICMP Address Mask reply.

This operating system fingerprinting method enables us to identify and distinguish between Sun Solaris, and HP-UX 10.30 &11.x operating systems to the other operating systems.

This method was tested against: Linux Kernel 2.4 test 2,4,5,6; Linux Kernel 2.2.x; FreeBSD 4.0, 3.4; OpenBSD 2.7,2.6; NetBSD 1.4.1,1.4.2; BSDI BSD/OS 4.0,3.1; Solaris 2.6,2.7,2.8; HP-UX 10.20, 11.0; Compaq Tru64 5.0; Aix 4.1,3.2; Irix 6.5.3, 6.5.8; Ultrix 4.2 – 4.5; OpenVMS v7.1-2; Novel Netware 5.1 SP1, 5.0, 3.12; Microsoft Windows 98/98SE, Microsoft Windows NT WRKS SP6a, Microsoft Windows NT Server SP4, Microsoft Windows 2000 Family.

7.2.5 DF Bit Echoing

Some operating systems, when receiving an ICMP query message with the DF bit set, will set the DF bit with their replies as well. Sometimes it will be in contrast with their regular behavior, which

will be not setting the DF Bit with their replies for a regular query that comes with the DF bit not set.

7.2.5.1 DF Bit Echoing with the ICMP Echo request

The `tcpdump` trace below illustrates an ICMP Echo request sent from a Linux based machine, using sing, to a Microsoft Windows 2000 Advanced Server based machine. The –G option with `sing` enable us to set the DF Bit with our requests:

```
[root@godfather /]# sing -echo -G -c 2 IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=0 DF! ttl=113 TOS=0 time=247.046 ms
16 bytes from IP_Address: icmp_seq=1 DF! ttl=113 TOS=0 time=260.024 ms
...

--- IP_Address sing statistics---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 230.024/246.194/260.024 ms
[root@godfather /]#
```

The `tcpdump` trace:

```
17:07:16.128308 if 4  > x.x.x.x > y.y.y.y: icmp: echo request (DF) (ttl
255, id 13170)
                        4500 0024 3372 4000 ff01 c846 xxxx xxxx
                        yyyy yyyy 0800 96e5 7d04 0000 14e7 bc39
                        11f5 0100
17:07:16.375256 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply (DF) (ttl
113, id 11936)
                        4500 0024 2ea0 4000 7101 5b19 yyyy yyyy
                        xxxx xxxx 0000 9ee5 7d04 0000 14e7 bc39
                        11f5 0100
```

Most of the operating systems that I have checked this behavior against acted the same as the Microsoft Windows 2000 Advanced Server. In the reply they produced, the DF bit was set.

Which operating systems are the exceptional and do not echo back the DF bit?
Linux based on Kernel 2.2.x, Ultrix v4.2 – 4.5, and Novell Netware.

How can we distinguish between these operating systems?

Since Linux and Ultrix are using an IP TTL field value of 255 in their ICMP query replies, and Novell Netware uses 128, we can divide the questionable IP addresses into two groups.

If we wish to further distinguish between the Linux based systems and the Ultrix based systems, we can send an ICMP Information request or an ICMP Address Mask request to the questioned IP addresses. The IP Addresses, which will produce a reply for our queries wll be those who are based on the Ultrix operating system.

7.2.5.2 DF Bit Echoing with the ICMP Address Mask request

With ICMP Address Mask requests we have a different story. Among the operating systems that answer for an ICMP Address Mask requests Sun Solaris, HPUX 11.x[57] & OpenVMS echo back the DF bit while Microsoft Windows 98, Microsoft Windows 98 SE, and Ultrix do not echo back the DF bit.

Again it is very simple to distinguish between the Microsoft Windows 98 family of operating systems and between the Ultrix based machines. This is since the Microsoft Windows 98 family is using 128 as their IP TTL field value in their ICMP query replies while Ultrix uses 255.

We have here a simple method to distinguish between Microsoft Windows 98 / 98 SE, and Ultrix machines to the rest of the operating systems world.

Another interesting piece of information is that the Microsoft Windows 98 family changed its behavior from DF echoing with the ICMP Echo request to not echoing the DF bit with ICMP Address Mask requests. This inconsistency is a factor with all Microsoft operating systems (Echoing with ICMP Echo request, not echoing with the other types of ICMP query).

7.2.5.3 DF Bit Echoing with the ICMP Timestamp request

Since a lot more operating systems answer for an ICMP Timestamp request than with the ICMP Address Mask request, we will have a bit more difficulty in identifying those.

Linux machines based on Kernel 2.2.x, Ultrix, Microsoft Windows 98/98SE/ME, and the Microsoft Windows 2000 Family will not echo back the DF bit with ICMP Timestamp replies they produce for corresponding ICMP Timestamp requests that sets their DF bit.

Here we can only distinguish between certain groups of operating systems; again it will be according to their IP TTL field value with their replies.

Linux would use 255 as its TTL field value for the ICMP Timestamp reply; Ultrix would use the same value. The Microsoft family of operating system members that will answer for this kind of query will use 128 as their IP TTL field value.

Again we have Linux and Ultrix on the one hand and certain members of the Microsoft based OSs family on the other hand.

7.2.5.4 Why this will work (for the skeptical)

All those skeptical will say that if they receive an ICMP query request with the DF bit set than it should be a clear sign that something is wrong and someone is probably trying to scan them. Think again. What will happen if a Sun Solaris / Linux Kernel 2.4.x / AIX 4.3 machine will query your machine? Than the same behavior will be produced.

This is an ICMP Echo request sent from a Sun Solaris 2.6 based machine to a Linux Kernel 2.2.14 based machine. We can see from the `snort` traces that the DF bit is set with the request and not set with the reply. But again if some one would mimic this behavior with a tool used on a Linux box to query the world, which is 100% mimicking a Sun Solaris request than we will never know if this is a legit request or an attempt for scanning / fingerprinting.

[57] When the PMTU Discovery process using ICMPEcho requests is not enabled, or when it works fast enough to set the DF bit with the first ICMP Address Mask reply.

```
08/10-23:32:52.201612 x.x.x.x -> y.y.y.y
ICMP TTL:239 TOS:0x0 ID:48656  DF
ID:2080   Seq:0  ECHO
39 93 10 A3 00 03 F0 E5 08 09 0A 0B 0C 0D 0E 0F  9...............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567


08/10-23:32:52.201649 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:349
ID:2080   Seq:0  ECHO REPLY
39 93 10 A3 00 03 F0 E5 08 09 0A 0B 0C 0D 0E 0F  9...............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

Operating System	Info. Request	Time Stamp Request	Address Mask Request	
Linux Kernel 2.4.x	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Linux Kernel 2.2.x	Not Answering	+ (- DF)	Not Answering	+ (- DF)
FreeBSD 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
FreeBSD 3.4	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.7	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.6	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
NetBSD	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 3.1	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Solaris 2.5.1	Not Answering			
Solaris 2.6	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.7	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.8	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	+ (+ DF)	+ (+ DF)
Compaq Tru64 v5.0		+ (+ DF)	Not Answering -	+ (+ DF)
Irix 6.5.3	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Irix 6.5.8	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
AIX 4.1		+ (+ DF)	Not Answering	+ (+ DF)
AIX 3.2		+ (+ DF)	Not Answering	+ (+ DF)
ULTRIX 4.2 – 4.5		+ (- DF)	+ (- DF)	+ (- DF)
OpenVMS v7.1-2		+ (+ DF)	+ (+ DF)	+ (+ DF)
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	+ (- DF)
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows 98 SE	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows ME	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		

Operating System		Time Stamp Request	Address Mask Request	Echo Request
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows 2000 Professional	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows 2000 Server	Not Answering	+ (- DF)	Not Answering	+ (+ DF)

Table 22: DF Bit Echoing

7.2.5.5 Combining all together

If we will combine all the information given in this section we can identify quite a lot of operating systems.

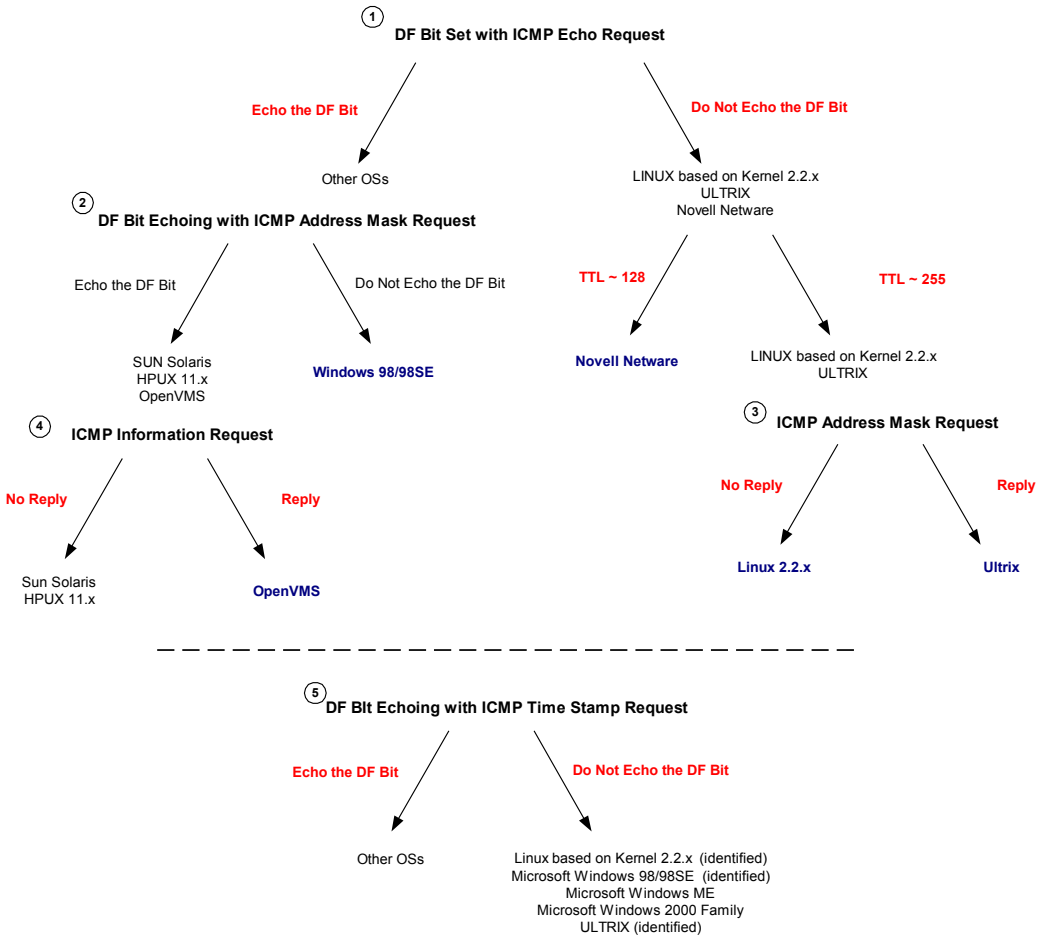


Diagram 6: An example of fingerprinting using the DF Bit Echoing technique

With the example above, we start our identification process with a query of ICMP Echo request with the DF bit set. Linux Kernel 2.2.x, Ultrix and Novell Netware will not echo back the DF bit. Since the original value assigned to the IP TTL field value with Novell Netware based machines is

128, and with Linux Kernel 2.2.x and Ultrix this value will be originally 255 we can divide the three operating systems into two groups. Our next step will be to query the questionable Linux Kernel 2.2.x and Ultrix IP addresses with an ICMP Address Mask requests. The IP addresses, which will answer, will be Ultrix based, while the non-answering IPs will be Linux Kernel 2.2.x based.

Now we will return to the IP addresses of the operating systems that did echo the DF in their replies (first step test). We will query them with an ICMP Address Mask request with the DF bit set.

From the operating systems that will answer the ICMP Address Mask query Microsoft Windows 98/98SE will not echo back the DF bit.

Sun Solaris, HPUX 11.x, and OpenVMS will echo back the DF bit with their ICMP Address Mask replies. We will use an ICMP Information request to divide this group of IP addresses. While the IP addresses of OpenVMS based machines will answer our query, Sun Solaris and HPUX 11.x based IP addresses will not answer the query.

7.2.6 Using Code field values different than zero within ICMP ECHO requests

An interesting detail I have discovered during my lab experiments for this research is when a wrong code is sent along with the correct type of ICMP query message, different operating systems will send different code values back.

In the next example I have sent an ICMP Echo request with the code field value set to 38 instead of 0, to a Linux machine running Linux Kernel 2.2.14.

We can examine at the `tcpdump` trace, the type and code fields are in bold type:

```
00:21:05.238649 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request (ttl 255,
id 13170)
                  4500 0024 3372 0000 ff01 08d3 xxxx xxxx
                  yyyy yyyy 0826 af13 2904 0000 41e4 c339
                  17a4 0300
00:21:05.485617 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 240, id
2322)
                  4500 0024 0912 0000 f001 4233 yyyy yyyy
                  xxxx xxxx 0026 b713 2904 0000 41e4 c339
                  17a4 0300
```

In the ICMP Echo reply the queried Linux Kernel 2.2.14 based machine have produced the code field value is set to 38 (decimal, 26 hex).

If we examine what RFC 792 requires, we see that Linux comply with it:

The sending side initializes the identifier (used to identify Echo requests aimed at different destination hosts) and sequence number (if multiple Echo requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP Echo request to the destination host. *In the ICMP header the code equals zero*. The recipient should *only change* the type to Echo reply and return the datagram to the sender.

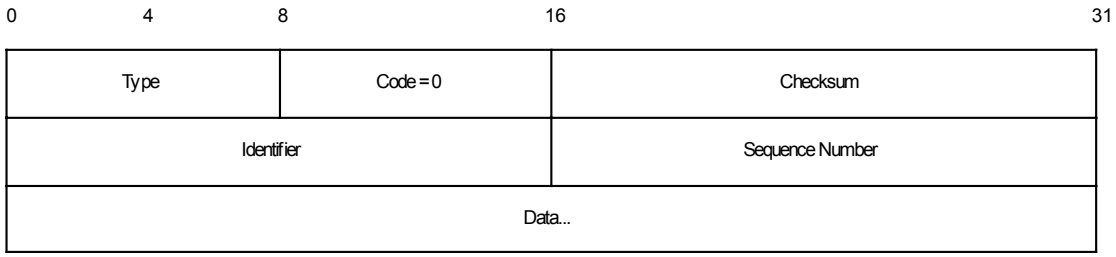


Figure 27: ICMP ECHO Request & Reply message format

This also means that we trust another machine to behave correctly, when that host produce the ICMP Echo reply.

Linux changes the type field value to 0 and sends the reply. The code field is unchanged.

The RFC does not outline what should happen if a host receives an ICMP query message with a wrong code. This might be because all ICMP query message types where defined with a default code, code 0.

I have checked the behavior of my Microsoft Windows 2000 Professional SP2 based machine. I have sent the same ICMP Echo Request message to the Microsoft Windows machine as I did with the previous example:

```
[root@godfather /root]# sing -echo -c 2 -x 26 172.18.2.200
SINGing to 172.18.2.200 (172.18.2.200): 16 data bytes
16 bytes from 172.18.2.200: seq=0 ttl=128 TOS=0 time=3.503 ms
16 bytes from 172.18.2.200: seq=1 ttl=128 TOS=0 time=2.949 ms

--- 172.18.2.200 sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.949/3.226/3.503 ms
[root@godfather /root]#
```

The snort trace:

```
05/21/01-15:32:25.765407  172.18.2.201 -> 172.18.2.200
ICMP TTL:255 TOS:0x0 ID:13170 IpLen:20 DgmLen:36
Type:8  Code:26  ID:7174   Seq:0  ECHO
D9 0A 09 3B 41 B9 0B 00                         ...;A...


05/21/01-15:32:25.765407  172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:18753 IpLen:20 DgmLen:36
Type:0  Code:0  ID:7174  Seq:0  ECHO REPLY
D9 0A 09 3B 41 B9 0B 00                         ...;A...
```

The Microsoft Windows 2000 Professional SP2 operating system changed the code field value on the ICMP Echo reply to the value of 0.

This method was tested with various operating systems including LINUX Kernel 2.4.x, IBM AIX 4.x & 3.2, SUN Solaris 2.51, 2.6, 2.7 & 2.8, OpenBSD 2.6 & 2.7, NetBSD 1.4.1, 1.4.2, BSDI BSD/OS 4.0 & 3.1, HP-UX 10.20 & 11.0, Compaq Tru64 v5.0, Irix 6.5.3 & 6.5.8, Ultrix 4.2-4.5, OpenVMS, FreeBSD 3.4, 4.0 & 4.1 and they produced the same results as the LINUX box (Kernel 2.2.x) did.

Microsoft Windows 4.0 Server SP4, Microsoft Windows NT 4.0 Workstation SP 6a, Microsoft Windows NT 4.0 Workstation SP3, Microsoft Windows 95 / 98 / 98 SE / ME have produced the same behavior as the Microsoft Windows 2000 Professional (Server & Advanced Server).

We have a fingerprinting method to differentiate between a Microsoft Windows based machine to the rest of the operating systems world using code values, which are different than zero, inside ICMP Echo requests.

7.2.7 Using Code field values different than zero within ICMP Timestamp Request

I have decided to map which operating systems will answer to an ICMP Timestamp request that will have its code field not set to zero, and how the ICMP Timestamp reply (if any) will help us identify those operating systems.

7.2.7.1 The non-answering Operating Systems

Interesting results were produced. The Microsoft Windows 98/98 SE/ME, and the Microsoft Windows 2000 family that have answered to ICMP Timestamp requests with the code field set to zero, now did not produce any reply back.

This enables us to group together certain versions of the Microsoft Windows operating systems.

7.2.7.2 Operating Systems the Zero out the Code field value on Reply

I was looking to see if there are operating systems in which answered the crafted ICMP Timestamp request with the Code field set to a value different than zero, which might zero out this field value with their ICMP Timestamp reply.

I have found that the Linux operating systems based on Kernel 2.2.x or on Kernel 2.4.x zero out the code field with ICMP Timestamp replies they produce for the corresponding ICMP Timestamp requests with the code field value that is different than zero.

The next example is an ICMP timestamp request sent from a Linux Kernel 2.2.14 based machine with the code field set to a value of 38 decimal / 26 hex. The targeted machine is a Linux Kernel 2.4 test 6 based machine. As we can see from the `tcpdump` trace, the targeted Linux Kernel 2.4 test 6 based machine zeroed out the code field with its ICMP Timestamp reply:

```
[root@godfather /root]# sing  -tstamp -x 38 -c 2 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=0 diff=24315927
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=0 diff=24316176

--- IP_Address sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
```

```
[root@godfather /root]#
```

The `tcpdump` trace:

```
20:10:18.138486 ppp0 > x.x.x.x > y.y.y.y: icmp: time stamp request (ttl
255, id 13170)
                  4500 0028 3372 0000 ff01 606c xxxx xxxx
                  yyyy yyyy 0d26 2e0c 7c04 0000 03af 451a
                  0000 0000 0000 0000
20:10:18.354222 ppp0 < y.y.y.y > x.x.x.x: icmp: time stamp reply (ttl
243, id 15717)
                  4500 0028 3d65 0000 f301 6279 yyyy yyyy
                  xxxx xxxx 0e00 888b 7c04 0000 03af 451a
                  0422 4e31 0422 4e31
```

7.2.7.3 Changed Patterns

The Linux Kernel 2.2.x/2.4.x operating system's behavior with the crafted ICMP Timestamp requests is in contrast with its behavior with the crafted ICMP Echo requests, both sent with the code field set to a value different than zero.

This also gives us a unique piece of information that enables us to identify Linux based machines.

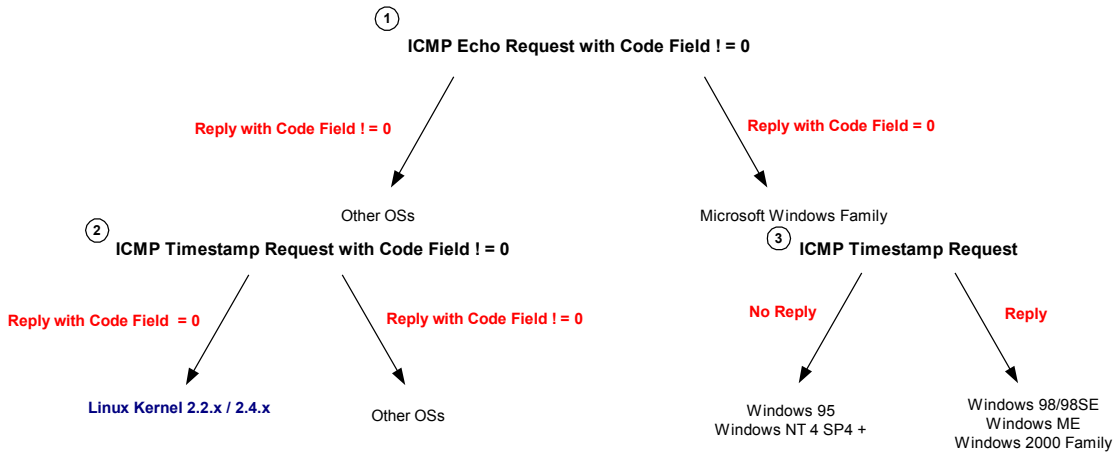


Diagram 7: An Example of Finger Printing Using crafted ICMP Echo & Timestamp Request

The diagram above describes a process in which we can use in order to differentiate between certain groups of operating systems.

The first step is sending an ICMP Echo request with the code field set to a value different than zero. The ICMP Echo replies with the code field equal to zero would distinguish the Microsoft based operating systems group, from the other operating systems.

Sending ICMP Timestamp requests with a code field value different than zero to the 'other OSs' group will identify Linux Kernel 2.2.x / 2.4 based machines (since they zero out the code field with their ICMP Timestamp replies).

Sending ICMP Timestamp request to the Microsoft Windows based group of operating systems will separate the group to those machines rather being windows 95 or windows NT 4 SP4 and above (not answer the query), to those that may be one of the following – Microsoft Windows 98 / SE / ME / Windows 2000 Family (answer the query).

7.3 Using ICMP Error Messages

7.3.1 Operating system, which do not generate ICMP Protocol Unreachable Error Messages

Several operating systems will not generate an ICMP Protocol Unreachable error message, when one is expected to be produced, in response to an offending datagram trying to use a protocol, which is not being used on those operating systems.

Those operating systems include:

- AIX
- DG-UX
- HP-UX

7.3.2 ICMP Error Message Quenching

RFC 1812 and RFC 1122 suggest limiting the rate at which various error messages are sent. Only few operating systems are known to follow this.

For example: An attacker can use this to send UDP packets to a random, high UDP port and count the number of ICMP Destination unreachable messages received within a given amount of time.

7.3.3 ICMP Error Message Quoting Size

Each ICMP error message includes the Internet Protocol (IP) Header and *at least* the first 8 data bytes of the datagram that triggered the error (the offending datagram); more than 8 bytes *may* be sent according to RFC 1122.

Most of the operating systems will quote the offending packets IP Header and the first 8 data bytes of the datagram that triggered the error. Several operating systems and networking devices will parse the RFC guidelines a bit different and will echo more than 8 bytes.

Which operating systems will quote more?
Linux based on Kernel 2.0.x/2.2.x/2.4.x, Sun Solaris, HPUX 11.x, MacOS 7.55/8.x/9.04, Nokia boxes, Foundry Switches (and other OSs and several Networking Devices) are a good example.

The fact is not new. Fyodor outlined this in his article “Remote OS Identification by TCP/IP Fingerprinting“[58].

The idea is in trying to differentiate between the different operating systems that quote more than the usual. How can this be done?

Looking for example at the amount of information quoted. Is there a limit to the quoted size? Will the quoted data be the entire offending packet or just part of it? Will the quoted data be quoted correctly? Will extra bytes be padded to the quoted data? and some other parameters.

The next example is with Sun Solaris 7. I have sent a UDP datagram to a closed UDP port:

```
00:13:35.559947 ppp0 > x.x.x.x.1084 > y.y.y.y.2000: udp 0 (ttl 64, id
44551)
                    4500 001c ae07 0000 4011 7aa4 xxxx xxxx
                    yyyy yyyy 043c 07d0 0008 a1ac

00:13:35.923691 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.1084 > y.y.y.y.2000: udp 0 (ttl 45,
id 44551) (DF) (ttl 236, id 63417)
                    4500 0038 f7b9 4000 ec01 44e5 yyyy yyyy
                    xxxx xxxx 0303 4f3c 0000 0000 4500 001c
                    ae07 0000 2d11 8da4 xxxx xxxx yyyy yyyy
                    043c 07d0 0008 a1ac
```

Please note that for having more than 8 data bytes quoted, you need to have data in the offending datagram. If not, there is nothing to quote beyond the regular 8 bytes (usually, if the OS is not padding other data bytes).

The next example is with Sun Solaris 8. I have sent a UDP datagram to a closed UDP port, adding 80 bytes of data to the datagram this time:

```
[root@godfather]# hping2 -2 -d 80 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 80 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The `tcpdump` trace:

```
11:52:50.830383 eth0 > x.x.x.x.2198 > y.y.y.y.0: udp 0 (ttl 64, id
17240)
                    4500 006c 4358 0000 4011 99ae xxxx xxxx
                    yyyy yyyy 0896 0000 0058 8b5f 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
```

[58] http://www.insecure.org/nmap/nmap-fingerprinting-article.html

```
                    5858 5858 5858 5858 5858 5858

11:52:51.367331 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2198 > y.y.y.y.0: udp 0 (ttl 48, id
17240) (DF) (ttl 231, id 49576)
                    4500 0070 c1a8 4000 e701 3469 yyyy yyyy
                    xxxx xxxx 0303 bf05 0000 0000 4500 006c
                    4358 0000 3011 a9ae xxxx xxxx yyyy yyyy
                    0896 0000 0058 8b5f 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
                    5858 5858 5858 5858 5858 5858 5858 5858
```

The result is an ICMP Port Unreachable Error message that will echo only 64 bytes of the offending datagram's data portion.

The limit of 64 bytes quoted from the offending packet's data portion is not limited to Sun Solaris only. HPUX 11.x, MacOS 7.55/8.x/9.04, will do the same by default.

Infact this is a tunnable parameter with Sun Solaris that can be changed using the `ndd` command. The parameter is *ip_icmp_return_data_bytes* and it is set by default to 64. You may change it to a value between 8 to 65536.

Other operating systems / networking devices will have their own limits. For example, Linux based on Kernel 2.2.x/2.4.x will send an ICMP Error Message up to 576 bytes long. Linux will quote 528 bytes from the data portion of the offending packet (576 minus 20 bytes of usuall IP Header, minus 8 bytes of the ICMP Header, minus the offending packet's IP Header that is 20 bytes will leave you with 528 bytes of data portion. This if no IP options are presented).

I know an operating system, and a family of networking devices that will pad extra data to the echoed offending packet. The Linux case is detailed in the next section. The next example is with Foundry Networks ServerIron running software version 07.1.02T12. I have sent a UDP datagram to a closed UDP port on the Foundry switch:

```
[root@godfather]# hping2 -2 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather]#
```

The `tcpdump` trace:

```
12:08:47.793503 eth0 > x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 64, id
44437)
                    4500 001c ad95 0000 4011 885f xxxx xxxx
                    yyyy yyyy 09c2 0000 0008 b13f
```

```
12:08:48.240208 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 51, id
44437) (ttl 51, id 17453)
                  4500 0044 442d 0000 3301 feaf yyyy yyyy
                  xxxx xxxx 0303 739c 0000 0000 4500 001c
                  ad95 0000 3311 955f xxxx xxxx yyyy yyyy
                  09c2 0000 0008 b13f dd2c 2a16 38e1 7646
                  7aaa 9d41
```

As it seems Foundry switches will pad 12 bytes with ICMP Port unreachable error messages.

Other fingerptinting facts that are outlined through this section will help us to differentiate between the operating systems, which carry the same behavior.

I have examined three ICMP Error Messages a Host can issue:

- ICMP Port Unreachable
- ICMP Protocol Unreachable
- ICMP Fragment Reassembly Time Exceeded

Other ICMP Error Messages, which a Host can issue and should be checked to see if they hold more fingerprinting differences, are:

- Source Quench
- Parameter Problem

7.3.4 LINUX ICMP Error Message Quoting Size Differences / The 20 Bytes from No Where

We must understand that there are differences between the different ICMP error messages, not only with their meaning, but also with their implementation. I was expecting that several characters with ICMP error messages will be the same with all of the ICMP error messages implemented in a certain operating system, but I was wrong regarding some operating systems.

The most interesting case is with the Linux operating system based on Kernel 2.2.x and 2.4.x.

The next example is with Linux based on Kernel 2.2.16 as the targeted machine, eliciting an ICMP Port Unreachable error message:

```
00:21:30.199408 ppp0 > x.x.x.x.2066 > y.y.y.y.2000: udp 0 (ttl 64, id
1732)
                  4500 001c 06c4 0000 4011 c895 xxxx xxxx
                  yyyy yyyy 0812 07d0 0008 4484

00:21:30.493691 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.2066 > y.y.y.y.2000: udp 0 (ttl 44,
id 1732) [tos 0xc0]  (ttl 238, id 53804)
                  45c0 0038 d22c 0000 ee01 4e60 yyyy yyyy
                  xxxx xxxx 0303 a88e 0000 0000 4500 001c
                  06c4 0000 2c11 dc95 xxxx xxxx yyyy yyyy
```

```
                    0812 07d0 0008 4484
```

The quoted data is the entire offending datagram. Linux ICMP error messages will be up to 576 bytes long according to the Linux source code.

The next example is with Linux Kernel 2.2.16 as the targeted operating system. With this example I have sent a protocol scan with `nmap`:

```
13:14:56.942897   < x.x.x.x > y.y.y.y: ip-proto-38 0 (ttl 39, id 37623)
                    4500 0014 92f7 0000 2726 02cb xxxx xxxx
                    yyyy yyyy
13:14:56.942964   > y.y.y.y > x.x.x.x: icmp: y.y.y.y protocol 38
unreachable Offending pkt: x.x.x.x > y.y.y.y: ip-proto-38 0 (ttl 39, id
37623) [tos 0xc0]  (ttl 255, id 1884)
                    45c0 0044 075c 0000 ff01 b59a yyyy yyyy
                    xxxx xxxx 0302 fb1a 0000 0000 4500 0014
                    92f7 0000 2726 02cb xxxx xxxx yyyy yyyy
                    0050 dc84 ae6f 6910 0000 0000 5004 0000
                    bd89 0000
```

inux added to the entire offending packet that was quoted, another 20 bytes.

Since Linux handles the ICMP Protocol Unreachable error messages like the ICMP Fragment Reassembly Time Exceeded error messages we will see the same pattern with ICMP Fragment Reassembly Time Exceeded error messages:

```
[root@godfather bin]# hping2 -c 1 -x -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y ppp0 y.y.y.y): NO FLAGS are set, 40 headers + 0 data
bytes

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather bin]#
```

The `tcpdump` trace:

```
19:49:22.999108 ppp0 > x.x.x.x.cvspserver > y.y.y.y.0: .
1709055398:1709055398(0) win 512 (frag 35247:20@0+) (DF) (ttl 64)
                    4500 0028 89af 6000 4006 e0ff xxxx xxxx
                    yyyy yyyy 0961 0000 65de 1da6 6a01 476b
                    5000 0200 bf71 0000

19:49:53.303196 ppp0 < y.y.y.y > x.x.x.x: icmp: ip reassembly time
exceeded Offending pkt: x.x.x.x.cvspserver > y.y.y.y.0: .
1709055398:1709055398(0) win 512 (frag 35247:20@0+) (DF) (ttl 45) [tos
0xc0]  (ttl 238, id 379)
                    45c0 0058 017b 0000 ee01 1a49 yyyy yyyy
```

```
xxxx xxxx 0b01 3caf 0000 0000 4500 0028
89af 6000 2d06 f3ff xxxx xxxx yyyy yyyy
0961 0000 65de 1da6 6a01 476b 5000 0200
bf71 0000 601d 1f0d 7a04 5045 0100 0000
4146 4345 4a45 4f46
```

Since Linux's ICMP error messages will not be bigger than 576 bytes long, if the offending packet will be big enough (not likely in real world situation) we will not see the added 20 bytes in the ICMP Fragment Reassembly / ICMP Protocol Unreachable error messages.

This unique pattern will allow us to identify Linux based machines even if the Precedence Bits value with the Linux ICMP Error messages will be changed to 0x000.

7.3.5 Foundry Networks Networking Devices Padded Bytes with ICMP Port Unreachable(s) / The 12 Bytes from No Where

Linux is not the only operating system that will have weird data bytes padded to one of its ICMP error messages.

Foundry Network's networking devices will pad extra 12 bytes of data with their ICMP Port Unreachable error messages. Our first example is with a ServerIron switch running software version 7.1.02T12, eliciting an ICMP Port Unreachable error message, for a UDP datagram trying to communicate with UDP port 0:

```
[root@godfather]# hping2 -2 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather]#
```

The `tcpdump` trace:

```
12:08:47.793503 eth0 > x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 64, id
44437)
                  4500 001c ad95 0000 4011 885f xxxx xxxx
                  yyyy yyyy 09c2 0000 0008 b13f

12:08:48.240208 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 51, id
44437) (ttl 51, id 17453)
                  4500 0044 442d 0000 3301 feaf yyyy yyyy
                  xxxx xxxx 0303 739c 0000 0000 4500 001c
                  ad95 0000 3311 955f xxxx xxxx yyyy yyyy
                  09c2 0000 0008 b13f dd2c 2a16 38e1 7646
                  7aaa 9d41
```

From the `tcpdump` trace we can conclude that the offending packet's IP header and the first 8 data bytes were quoted correctly. Right after these, 12 bytes were padded, that came from nowhere.

The next example is with Foundry Network's BigIron 8000 running software version 6.6.05T51. With this test I have sent a UDP datagram with 80 bytes of data to a closed UDP port (UDP port 80) on the BigIron 8000:

```
[root@godfather /root]# hping2 -2 -c 3 -d 80 y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y  (ppp0 y.y.y.y  ): udp mode set, 28 headers + 80 data
bytes
ICMP Port Unreachable from y.y.y.y  (y.y.y.y)
ICMP Port Unreachable from y.y.y.y  (y.y.y.y)
ICMP Port Unreachable from y.y.y.y  (y.y.y.y)

--- y.y.y.y  hping statistic ---
3 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#
```

The `tcpdump` trace:

```
11:40:36.694235 ppp0 > x.x.x.x.2779 > y.y.y.y.0: udp 80 (ttl 64, id
25211)
                  4500 006c 627b 0000 4011 2e7a xxxx xxxx
                  yyyy yyyy 0adb 0000 0058 3d09 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858

11:40:37.913018 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2779 > y.y.y.y.0: udp 80 (ttl 52, id
25211) (ttl 52, id 60504)
                  4500 0044 ec58 0000 3401 b0d4 yyyy yyyy
                  xxxx xxxx 0303 edf3 0000 0000 4500 006c
                  627b 0000 3411 3a7a xxxx xxxx yyyy yyyy
                  0adb 0000 0058 3d09 1c1d 1e1f 2021 2223
                  2425 2627
```

Again, the offending packet's IP Header and the first 8 data bytes are quoted correctly. 12 data bytes are padded right after.

A nice pattern that allows us to identify Foundry Network's networking devices.

7.3.6 ICMP Error Message Echoing Integrity

Tested with ICMP Port Unreachable Error Message

When sending back an ICMP error message, some stack implementations may alter the original IP header, which is echoed back with the ICMP error message.

If a malicious computer attacker examines the types of alternation that have been made to the headers, he may be able to make certain assumptions about the target operating system.

The only two field values we expect to be changed are the IP time-to-live field value and the IP header checksum. The IP TTL field value changes because the field is decreased by one, each time the IP Header is being processed. The IP header checksum is recalculated each time the IP TTL field value is descreased.

Fyodor gives the following examples in his article "Remote OS detection via TCP/IP Stack Finger Printing"[59]:

> "For example, AIX and BSDI send back an IP 'total length' field that is 20 bytes too high. Some BSDI, FreeBSD, OpenBSD, ULTRIX, and VAXen change the IP ID that you sent them. While the checksum is going to change due to the changed TTL anyway, there are some machines (AIX, FreeBSD, etc.) which send back an inconsistent or 0 checksum. Same thing goes with the UDP checksum."

This section deals with the ICMP Port Unreachable error message.

7.3.6.1 AIX 4.2.1, 4.3, 4.3 fix pack 2

In the next example I have sent a UDP datagram to a closed UDP port on an AIX 4.3 based machine using the `hping2` utility. This is the `tcpdump` trace:

```
12:33:17.319275 ppp0 > x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 47349)
                  4510 001c b8f5 0000 4011 9bea xxxx xxxx
                  yyyy yyyy 0870 0000 0008 d18c

12:33:17.614823 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 49, id 47349, bad cksum aaea!) [tos 0x10]  (ttl 241, id 17965)
                  4510 0038 462d 0000 f101 5da6 yyyy yyyy
                  xxxx xxxx 0303 f470 0000 0000 4510 0030
                  b8f5 0000 3111 aaea xxxx xxxx yyyy yyyy
                  0870 0000 0008 0000
```

Several changed were made to the offending packet's data when echoed:

- **IP Total Length Field -** The total length field with the original UDP datagram equal to 28 (*001c hex*) bytes. With the echoed offending packet's IP header this value was changed to 48 (*0030 hex*) bytes. *20 bytes more than the original UDP datagram's length*.

[59] http://www.insecure.org/nmap/nmap-fingerprinting-article.html

- **IP TTL Field value -** With the ICMP error message this value is set to the value, which reached its final destination (with this example the targeted host). When it reached it target the TTL was set to 49. We also learn the target is 64-49 = 15 hops away.
- **IP Header Checksum -** The IP Header checksum was changed because the IP Total Length field value and the IP TTL field value were changed.
- **UDP Header Checksum** – The UDP header checksum with the echoed information equal to zero.

7.3.6.2 AIX 4.1

In the next example I have sent a UDP datagram to a closed UDP port on an AIX 4.1 based machine using the `hping2` utility. This is the `tcpdump` trace:

```
00:56:07.894612 ppp0 > x.x.x.x.1594 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 2153)
                  4508 001c 0869 0000 4011 c54f xxxx xxxx
                  yyyy yyyy 063a 0000 0008 4c93

00:56:08.204551 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.1594 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 47, id 2153, bad cksum d64f!) [tos 0x8]  (ttl 239, id 1065)
                  4508 0038 0429 0000 ef01 1a83 yyyy yyyy
                  xxxx xxxx 0303 aa13 0000 0000 4508 0030
                  0869 0000 2f11 d64f xxxx xxxx yyyy yyyy
                  063a 0000 0008 4c93
```

Several changed were made to the offending packet's data when echoed:

- **IP Total Length Field -** The total length field with the original UDP datagram equal to 28 bytes. With the echoed original IP header this value was changed to 48 bytes. 20 bytes more than the original UDP datagram's length.
- **IP TTL Field value -** With the ICMP error message this value is set to the value, which reached its final destination (with this example the targeted host). When it reached it target the TTL was set to 47. We also learn the target is 64-47 = 17 hops away.
- **IP Header Checksum -** The IP Header checksum was changed because the IP Total Length field value and the IP TTL field value were changed.

7.3.6.2.1 ICMP Error Message Echoing Integrity with different 4.x versions of AIX

In contrast to AIX version 4.3 and 4.2.1 AIX version 4.1 use the original UDP Checksum. This detail helps us to differentiate between the different versions of AIX.

7.3.6.3 BSDI 4.x

In the next example I have sent, again, a UDP datagram to a close UDP port, this time on a BSDI 4.1 based machine. The following is the `tcpdump` trace:

```
01:01:11.128420 ppp0 > x.x.x.x.2933 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 49317)
                  4508 001c c0a5 0000 4011 9209 xxxx xxxx
```

```
                    yyyy yyyy 0b75 0000 0008 cc4e

01:01:11.484552 ppp0 < y.y.y.y.4 > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2933 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 53, id 49317, bad cksum 0!) (ttl 242, id 16127)
                    4500 0038 3eff 0000 f201 61ab yyyy yyyy
                    xxxx xxxx 0303 c226 0000 0000 4508 0030
                    c0a5 0000 3511 0000 xxxx xxxx yyyy yyyy
                    0b75 0000 0008 cc4e
```

Again several changed were made to the offending packet's IP Header when echoed:

- **IP Total length -** With the echoed IP Header this field value was changed from the original 28 bytes to 48 bytes. 20 bytes more than the original.
- **IP TTL Field Value** – Changed according to the hop count. Was equal to 53 when arrived to its destination. The target is 64 – 53 = 11 hops away.
- **IP Header Checksum** – Changed, and with the ICMP error is now *equal to zero!*

7.3.6.4 FreeBSD 3.x up to 4.1.1 (not including)

The next example is with FreeBSD 4.1:

```
00:52:19.055758 ppp0 > x.x.x.x.1393 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 58965)
                    4508 001c e655 0000 4011 3f63 xxxx xxxx
                    yyyy yyyy 0571 0000 0008 a55c

00:52:19.464548 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.1393 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 47, id 21990, bad cksum 5063!) (ttl 238, id 27639)
                    4500 0038 6bf7 0000 ee01 0bbd yyyy yyyy
                    xxxx xxxx 0303 87f3 0000 0000 4508 001c
                    55e6 0000 2f11 5063 xxxx xxxx yyyy yyyy
                    0571 0000 0008 0000
```

Several changed were made to the offending packet's data when echoed:

- The **IP Identification** field value is changed. This field is constructed with 16bit. The first 8 bits changed places with the second pair of 8 bits constructing this field. With the original datagram this field value was e655, with the echoed IP header it is 55e6[60].
- The **IP TTL field value** has changed. The target is 64 – 47 = 17 hops away.
- The **IP Header Checksum** has changed because some of the parameters were changed as well. We can name the IP TTL field value and the IP Total Length field value as an example.
- The **UDP checksum** is changed and now it *equal to zero!*

[60] http://www.freebsd.org/cgi/query-pr.cgi?pr=16240 ; Patches were issued. This is fixed with FreeBSD 4.1.1.

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
Linux Kernel 2.4.x	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Linux Kernel 2.2.x	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
FreeBSD 4.0	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
FreeBSD 4.11	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
BSDI 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed. Now equals to ZERO!	Same
Sun Solaris 2.6	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.7	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.8[61]	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
HPUX 11.0	No -> Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Compaq Tru64	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!

[61] The DF Bit is set.

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
DG-UX 5.6	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.3 fp2, 4.3, 4.2.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Same
ULTRIX	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
OpenVMS	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
Microsoft windows 98						
Mirosoft Windows 98SE	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows ME	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows NT 4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows 2000 Family	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same

Table 23: ICMP Error Message Echoing Integrity

7.3.7 Novell Netware Echoing Integrity Bug with ICMP Fragment Reassembly Time Exceeded

Novell Netware operating systems have a unique pattern with ICMP Fragment Reassembly Time Exceeded error messages they produce.

In general, when an ICMP error message is produced, the offending packet's IP Header + at least 8 bytes of data are quoted with the error message.

If we examine closely the next example, we can see that the offending packet's IP TTL field value echoed back is zero.

We expect this value to decrease from the value initially assigned, but not to be zero. Since this value should change from one hop to another, the checksum need to be recalculated each time. With the Novell Netware error message we can see that the checksum echoed is miscalculated.

...And again this is a Fragment Reassembly Time Exceeded ICMP error message and not an ICMP Time Exceeded in Transit error message.

The next example is with Novell Netware 5.1:

```
[root@godfather bin]# hping2 -c 1 -x -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): NO FLAGS are set, 40 headers + 0 data
bytes

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather bin]#
```

The `tcpdump` trace:

```
20:12:28.008893 ppp0 > x.x.x.x.1865 > y.y.y.y.0: .
687160929:687160929(0) win 512 (frag 58586:20@0+) (DF) (ttl 64)
                        4500 0028 e4da 6000 4006 c236 xxxx xxxx
                        yyyy yyyy 0749 0000 28f5 3e61 669e 9f15
                        5000 0200 c5d2 0000

20:12:41.313202 ppp0 < y.y.y.y > x.x.x.x: icmp: ip reassembly time
exceeded Offending pkt: [|tcp] (frag 58586:20@0+) (DF) [ttl 0] (bad
cksum d336!) (ttl 111, id 9591)
                        4500 0038 2577 0000 6f01 b28f yyyy yyyy
                        xxxx xxxx 0b01 b55f 0000 0000 4500 0028
                        e4da 6000 0006 d336 xxxx xxxx yyyy yyyy
                        0749 0000 28f5 3e61
```

This unique pattern enables us to determine if the operating system in question is a Novell Netware or other with one datagram only.

7.3.8 The Precedence bits with ICMP Error Messages

Identifying Linux Based Machines

Each IP Datagram has an 8-bit field called the "TOS Byte", which represents the IP support for prioritization and Type-of-Service handling.

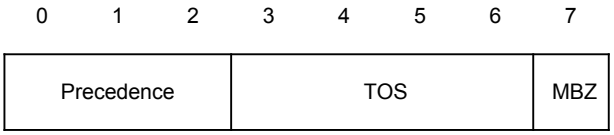


Figure 28: The Type of Service Byte

The "TOS Byte" consists of three fields.

The "Precedence field", which is 3-bit long, is intended to prioritize the IP Datagram. It has eight levels of prioritization[62]:

Precedence	Definition
0	Routine (Normal)
1	Priority
2	Immediate
3	Flash
4	Flash Override
5	Critical
6	Internetwork Control
7	Network control

Table 24: Precedence Field Values

Higher priority traffic should be sent before lower priority traffic.

The second field, 4 bits long, is the "Type-of-Service" field. It is intended to describe how the network should make tradeoffs between throughput, delay, reliability, and cost in routing an IP Datagram.

The last field, the "MBZ" (most be zero), is unused and most be zero. Routers and hosts ignore this last field. This field is 1 bit long.

RFC 1122 Requirements for Internet Hosts -- Communication Layers, states:
"The Precedence field is intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of this document and the IP standard specification. Vendors should consult the Defense Communication Agency (DCA) for guidance on the IP Precedence field and its implications for other protocol layers. However, vendors should note that the use of precedence will most likely require that its value be passed between protocol layers in just the same way as the TOS field is passed".

[62] RFC 791 – Internet Protocol, http://www.ietf.org/rfc/rfc791.txt.

Other precedence information is available with RFC 1812 Requirements for IP Version 4 Routers: "4.3.2.5 TOS and Precedence
…

ICMP Source Quench error messages, if sent at all, MUST have their IP Precedence field set to the **same value as the IP Precedence field in the packet that provoked the sending of the ICMP Source Quench message**. All **other ICMP error messages** (Destination Unreachable, Redirect, Time Exceeded, and Parameter Problem) **SHOULD** have their precedence value set to **6** (INTERNETWORK CONTROL) or **7** (NETWORK CONTROL). The IP Precedence value for these error messages MAY be settable".

With the operating systems I have checked, nearly all used the value of 0x000 for the Precedence bits field with ICMP error messages.

All but Linux.

Fyodor had outlined in his paper "Remote OS Identification by TCP/IP Fingerprinting" [63] the fact that Linux is using the value of 0xc0 (an unused precedence value) as its TOS byte value with ICMP Port Unreachable error messages.

In the next example we have sent one UDP packet destined to port 50 (which is closed on the destination machine) from one Linux machine to another, both running Linux Kernel 2.2.16:

```
[root@stan /root]# hping2 -2 192.168.5.5 -p 50 -c 1
default routing not present
HPING 192.168.5.5 (eth0 192.168.5.5): udp mode set, 28 headers + 0 data
bytes
ICMP Port Unreachable from 192.168.5.5  (kenny.sys-security.com)

--- 192.168.5.5 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The `snort` trace:

```
03/12-12:54:47.274096 192.168.5.1:2420 -> 192.168.5.5:50
UDP TTL:64 TOS:0x0 ID:57254
Len: 8

03/12-12:54:47.274360 192.168.5.5 -> 192.168.5.1
ICMP TTL:255 TOS:0xC0 ID:0
DESTINATION UNREACHABLE: PORT UNREACHABLE
00 00 00 00 45 00 00 1C DF A6 00 00 40 11 0F D4  ....E.......@...
C0 A8 05 01 C0 A8 05 05 09 74 00 32 00 08 6A E1  .........t.2..j.
```

This abnormality with Linux is not only limited to ICMP Destination Unreachable Port Unreachable error messages.

[63] This fact was discovered by Fyodor. http://www.insecure.org/nmap/nmap-fingerprinting-article.html

Lets examine the next `tcpdump` trace:

```
00:30:08.339498 < x.x.x.x > y.y.y.y: ip-proto-72 0 (ttl 49, id 38624)
                  4500 0014 96e0 0000 3148 f4bf xxxx xxxx
                  yyyy yyyy

00:30:08.339559 > y.y.y.y > x.x.x.x: icmp: y.y.y.y protocol 72
unreachable Offending pkt: x.x.x.x > y.y.y.y: ip-proto-72 0 (ttl 49, id
38624) [tos 0xc0]  (ttl 255, id 37)
                  45c0 0044 0025 0000 ff01 bcd1 yyyy yyyy
                  xxxx xxxx 0302 fb1a 0000 0000 4500 0014
                  96e0 0000 3148 f4bf xxxx xxxx yyyy yyyy
                  0050 d909 621b 96f7 0000 0000 5004 0000
                  df71 0000
```

The ICMP error message produced by a Linux machine based on Kernel 2.2.14, is Destination Unreachable Protocol Unreachable (Type 3 Code 2). As it can be seen the TOS Byte value that was used is again 0xc0. Which is an unused Precedence bits value.

Linux embraced the behavior RFC 1812 suggested and sends all his ICMP error messages with the Precedence field value sent to 0xc0 (value of 6).

Just to remind the reader – Linux is not a router.

7.3.9 TOS Bits (=field) Echoing with ICMP Error

Identifying AIX 4.x, DGUX, and Linux Kernel 2.2.x / 2.4.x

RFC 1394 specify that an ICMP error message be always sent with the default TOS field value of 0000 (TOS field=TOS bits in the TOS Byte).

When an offending packet with a TOS field value of 0x0000 is eliciting an ICMP error message from an offended host, the TOS field value with all the operating systems I have checked will be set to 0x0000.

If we will pay attention to the TOS Byte we will see that Linux and several routers will use the value of 0xc0 for the precedence field.

What will happen if the TOS field with the offending packet will be set to a value different than the default (0x0000)?

We will have several operating systems that will echo the TOS field back with the ICMP error message.

Our first example is with an AIX 4.3 machine, where a UDP datagram is sent with a TOS field value of 0x10 hex:

```
12:33:17.319275 ppp0 > x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 47349)
                  4510 001c b8f5 0000 4011 9bea xxxx xxxx
                  yyyy yyyy 0870 0000 0008 d18c
```

```
12:33:17.614823 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 49, id 47349, bad cksum aaea!) [tos 0x10]  (ttl 241, id 17965)
                  4510 0038 462d 0000 f101 5da6 yyyy yyyy
                  xxxx xxxx 0303 f470 0000 0000 4510 0030
                  b8f5 0000 3111 aaea xxxx xxxx yyyy yyyy
                  0870 0000 0008 0000
```

As it can be seen from the trace, the TOS field value was echoed back by the AIX based machine. This was tested against AIX 4.1, 4.2.1, 4.3, 4.3 fix pack2.

The next example is with DGUX 5.6:

```
12:58:57.663517 ppp0 > x.x.x.x.1074 > y.y.y.y.11: udp 0 [tos 0x8]  (ttl
64, id 47314)
                  4508 001c b8d2 0000 4011 a037 xxxx xxxx
                  yyyy yyyy 0432 000b 0008 d9e1

12:58:57.984820 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 11
unreachable Offending pkt: x.x.x.x.1074 > y.y.y.y.11: udp 0 [tos 0x8]
(ttl 52, id 47314) [tos 0x8]  (ttl 52, id 16984)
                  4508 0038 4258 0000 3401 22a6 yyyy yyyy
                  d508 c41c 0303 f8b7 0000 0000 4508 001c
                  b8d2 0000 3411 ac37 xxxx xxxx yyyy yyyy
                  0432 000b 0008 0000
```

How can we differentiate between DGUX and AIX? If we will pay attention to the echoing integrity. AIX 4.x sets the IP total length field value, with the echoed offending IP Header, to a value 20 bytes longer than the original. DGUX quote this field value correctly.

The last operating system, which I have found echoing the TOS field value with its ICMP error messages, is Linux operating systems based on Kernel 2.2.x & 2.4 (the versions of the Kernel that I have tested):

```
00:50:43.759906 ppp0 > x.x.x.x.1952 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 15952)
                  4510 001c 3e50 0000 4011 e6b2 xxxx xxxx
                  yyyy yyyy 07a0 0000 0008 a27f

00:50:44.154556 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.1952 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 47, id 15952) [tos 0xd0]  (ttl 238, id 54662)
                  45d0 0038 d586 0000 ee01 a0af yyyy yyyy
                  xxxx xxxx 0303 52d5 0000 0000 4510 001c
                  3e50 0000 2f11 f7b2 xxxx xxxx yyyy yyyy
                  07a0 0000 0008 a27f
```

Another unique pattern with Linux is setting the Precedence bits field value to 0xc0 with ICMP error messages. This helps us to differentiate Linux from the other operating systems that echo the TOS field value.

While Linux embraced RFC 1812 instructions for routers regarding the TOS and Precedence fields, the other operating systems that echoed the TOS field value didn't seem to have a good excuse for doing so.

7.3.10 DF Bit Echoing with ICMP Error Messages

We already have the DF Bit Echoing method with ICMP query message types (& Replies); I was thinking why this couldn't happen with ICMP error messages as well?

What will happen if we will set the DF bit with an offending packet that will generate an ICMP error message? Will the DF Bit be set with the ICMP error message?

In the next example, a UDP datagram is sent to a closed UDP port, to elicit an ICMP Port Unreachable error message. The DF bit is set with the offending datagram. As it can be seen the DF bit is set with the ICMP error message the FreeBSD 4.1.1 machine, which was the target system issued back.

```
[root@godfather /root]# hping2 -2 -p 2000 -c 2 -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y  (host_address)
ICMP Port Unreachable from y.y.y.y  (host_address)

--- y.y.y.y hping statistic ---
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#
```

The `tcpdump` trace:

```
00:31:29.805075 ppp0 > x.x.x.x.1403 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 19417)
                  4500 001c 4bd9 4000 4011 452b xxxx xxxx
                  yyyy yyyy 057b 07d0 0008 48c6

00:31:30.103692 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.1403 > y.y.y.y.2000: udp 0 (DF) (ttl
45, id 19417) (DF) (ttl 238, id 47017)
                  4500 0038 b7a9 4000 ee01 2b4e yyyy yyyy
                  xxxx xxxx 0303 efa9 0000 0000 4500 001c
                  4bd9 4000 2d11 582b xxxx xxxx yyyy yyyy
                  057b 07d0 0008 0000
```

We can distinguish between the group of operating systems, which will echo back the DF bit with their replies, to the group of operating systems that will not.

The next example is with Microsoft Windows ME:

```
00:49:45.853751 ppp0 > x.x.x.x.1580 > y.y.y.y.10: udp 0 (DF) (ttl 64,
id 63227)
                    4500 001c f6fb 4000 4011 730a xxxx xxxx
                    yyyy yyyy 062c 000a 0008 28dd

00:49:46.173681 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 10
unreachable Offending pkt: x.x.x.x.1580 > y.y.y.y.10: udp 0 (DF) (ttl
55, id 63227) (ttl 119, id 430)
                    4500 0038 01ae 0000 7701 714c yyyy yyyy
                    xxxx xxxx 0303 cde1 0000 0000 4500 001c
                    f6fb 4000 3711 7c0a xxxx xxxx yyyy yyyy
                    062c 000a 0008 28dd
```

Among the operating systems I have checked Linux machines based on Kernel 2.2.x / 2.4.x, ULTRIX, Novell Netware, and Microsoft Windows 98/98SE/ME/NT4SP6A/Windows 2000 Family, will not echo back the DF bit with their ICMP Error messages.

How can we distinguish between the operating systems in the non-DF echoing group?
Since Linux is using the value of 0xc0 hex for his Precedence Bits field value for all ICMP error messages we can separate it instantly.

```
00:25:17.203727 ppp0 > x.x.x.x.1421 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 11969)
                    4500 001c 2ec1 4000 4011 b938 xxxx xxxx
                    yyyy yyyy 058d 07d0 0008 9fa9

00:25:17.573698 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.1421 > y.y.y.y.2000: udp 0 (DF) (ttl
45, id 11969) [tos 0xc0]  (ttl 236, id 38250)
                    45c0 0038 956a 0000 ec01 e5c2 yyyy yyyy
                    xxxx xxxx 0303 4fee 0000 0000 4500 001c
                    2ec1 4000 2d11 cc38 xxxx xxxx yyyy yyyy
                    058d 07d0 0008 9fa9
```

ULTRIX echo integrity is not that good. The offending packet echoing will set both the IP Header Checksum and the Original UDP Checksum to zero. It will also miscalculate the IP ID field value and will flip the first 8 bits with the second one, creating a false value for it:

```
00:29:05.013726 ppp0 > x.x.x.x.1188 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 34921)
                    4500 001c 8869 4000 4011 5f85 xxxx xxxx
                    yyyy yyyy 04a4 07d0 0008 a087

00:29:05.383686 ppp0 < 194.47.250.222 > x.x.x.x: icmp: y.y.y.y udp port
2000 unreachable Offending pkt: x.x.x.x.1188 > y.y.y.y.2000: udp 0 (ttl
45, id 27016, bad cksum 0!) (ttl 236, id 9736)
                    4500 0038 2608 0000 ec01 55da yyyy yyyy
                    xxxx xxxx 0303 c1e7 0000 0000 4500 001c
                    6988 0000 2d11 0000 xxxx xxxx yyyy yyyy
```

```
04a4 07d0 0008 0000
```

This will leave us with Novell Netware and the various Microsoft Windows Operating Systems.

As discussed in the section dealing with "*Novell Netware Echoing Integrity Bug with ICMP Fragment Reassembly Time Exceeded* ", when a Novell Netware operating system issues an ICMP Time Exceeded error message it will zero out the IP TTL field value with the echoed offending packet. We will use this fingeprinting technique and send a fragment of a packet to the questioned IP addresses that will elicit an ICMP Time Exceeded error messages.

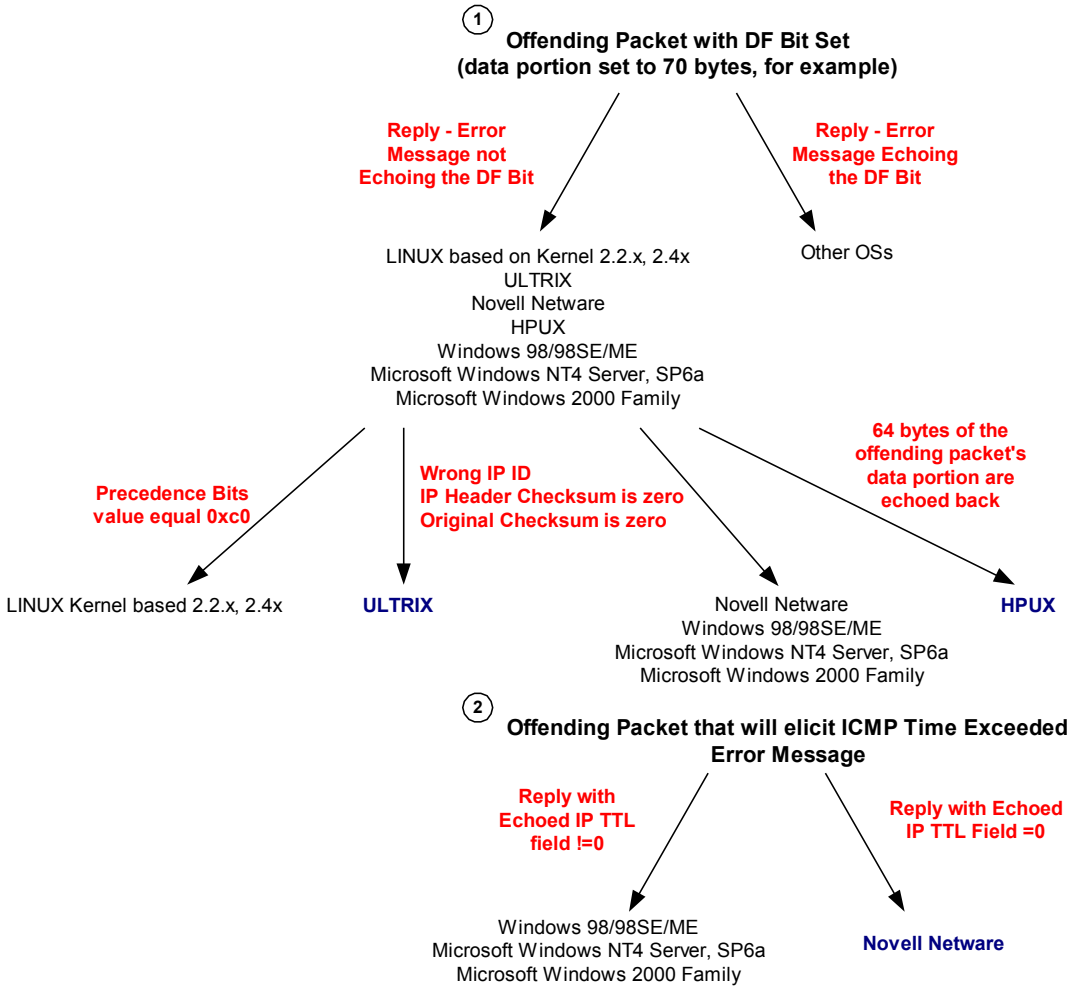


Diagram 8: DF Bit Echoing with ICMP Error Messages

We can take a second approach using the ICMP Fragment Reassembly Time Exceeded error messages. We will send an offending packet with the DF bit set that will elicit an ICMP Fragment Reassembly Time Exceeded error message back from the targeted IP addresses. Novell Netware, Linux based Kernel 2.2.x and 2.4x, and the various Microsoft Windows operating systems will set the DF bit with their replies. Linux and Novell have their unique fingerprinting with ICMP Fragment Reassembly Time Exceeded error messages, enabling us to isolate the Microsoft based operating systems based machines.

HP-UX 11.x based machines will have a unique behavior when the PMTU discovery process based on ICMP Echo Requests is enabled (by default). In the next example I have sent a UDP datagram to port 53 (DNS) of the targeted HPUX machine.

```
[root@godfather /root]# hping2 -2 -p 53 -c 2 -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (unknown host name)
ICMP Port Unreachable from y.y.y.y (unknown host name)

--- y.y.y.y hping statistic ---
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#
```

The tcpdump trace:

```
00:45:02.490445 ppp0 > x.x.x.x.codasrv > y.y.y.y.domain: 0 [0q] (0)
(DF) (ttl 64, id 7454)
                         4500 001c 1d1e 4000 4011 e708 xxxx xxxx
                         yyyy yyyy 0980 0035 0008 bf7e
```

As an instant reply the PMTU discovery process, which is based upon ICMP Echo request(s), is started:

```
00:45:03.113686 ppp0 < y.y.y.y > x.x.x.x: icmp: echo request (DF) (ttl
242, id 25153)
                         4500 05dc 6241 4000 f201 ea34 yyyy yyyy
                         xxxx xxxx 0800 7e52 9abc def0 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

My Linux based machine replied with ICMP Echo reply:

```
00:45:03.113787 ppp0 > x.x.x.x > y.y.y.y: icmp: echo reply (ttl 255, id
98)
                         4500 05dc 0062 0000 ff01 7f14 xxxx xxxx
                         yyyy yyyy 0000 8652 9abc def0 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000
```

The first ICMP Port Unreachable error message arrives without the DF bit set:

```
00:45:03.123692 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port domain
unreachable Offending pkt: x.x.x.x.codasrv > y.y.y.y.domain: 0 [0q] (0)
(DF) (ttl 51, id 7454) (ttl 242, id 25154)
                    4500 0038 6242 0000 f201 2fd8 yyyy yyyy
                    xxxx xxxx 0303 33c1 0000 0000 4500 001c
                    1d1e 4000 3311 f408 xxxx xxxx yyyy yyyy
                    0980 0035 0008 bf7e
```

A second UDP datagram is sent:

```
00:45:03.493752 ppp0 > x.x.x.x.codasrv-se > y.y.y.y.domain: 56810+ (0)
(DF) (ttl 64, id 59904)
                    4500 001c ea00 4000 4011 1a26 xxxx xxxx
                    yyyy yyyy 0981 0035 0008 bf7d
```

The ICMP Port Unreachable error message that was sent for the second UDP datagram now sets the DF bit as part of the PMTU discovery process maintenance:

```
00:45:03.813687 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port domain
unreachable Offending pkt: x.x.x.x.codasrv-se > y.y.y.y.domain: 26990
op5+ [b2&3=0x2d61] [29188a] [25700q] [24946n] [28769au] (0) (DF) (ttl
51, id 59904) (DF) (ttl 242, id 25155)
                    4500 0038 6243 4000 f201 efd6 yyyy yyyy
                    xxxx xxxx 0303 33c1 0000 0000 4500 001c
                    ea00 4000 3311 2726 xxxx xxxx yyyy yyyy
                    0981 0035 0008 bf7d
```

If you are sending only one offending datagram to the targeted HPUX 11.x based machine, you might not see the change in pattern (but you will still receive an ICMP Echo request 'the HPUX style' from the targeted host).

So how can we distinguish HPUX from the other operating systems?

HPUX based operating system(s) machines will echo up to 64 bytes of the offending packet's data portion. By sending a bigger offending datagram (for example with 80 bytes of data portion) we can examine which of the operating systems in question, which do not set the DF bit with the ICMP error message, will echo 64 bytes of the data portion (or an OS that will echo more than 8 data bytes and will not set the the precedence bits to 0xc0).

When the PMTU discovery process based on ICMP Echo Requests will not be enabled than we will see the following pattern:

```
[root@godfather /]# hping2 -2 -c 2 -y 172.18.1.5
eth0 default routing interface selected (according to /proc)
HPING 172.18.1.5 (eth0 172.18.1.5): udp mode set, 28 headers + 0 data
bytes
ICMP Port Unreachable from 172.18.1.5  (unknown host name)
ICMP Port Unreachable from 172.18.1.5  (unknown host name)

--- 172.18.1.5 hping statistic ---
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /]#
```

The `snort` trace:

```
05/29/01-18:29:39.370930 172.18.2.201:1749 -> 172.18.1.5:0
UDP TTL:64 TOS:0x0 ID:32891 IpLen:20 DgmLen:28 DF
Len: 8


05/29/01-18:29:39.371132 172.18.1.5 -> 172.18.2.201
ICMP TTL:254 TOS:0x0 ID:31414 IpLen:20 DgmLen:56 DF
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.201:1749 -> 172.18.1.5:0
UDP TTL:63 TOS:0x0 ID:32891 IpLen:20 DgmLen:28
Len: 8
** END OF DUMP
00 00 00 00 45 00 00 1C 80 7B 40 00 3F 11 5F 63  ....E....{@.?._c
AC 12 02 C9 AC 12 01 05 06 D5 00 00 00 08 9D 16  ................
```

7.4 Not that useful fingerprinting method(s)

7.4.1 Unusual Big ICMP Echo Request

What will happen if we will send an unusual big ICMP echo request message that will require its fragmentation? Will the queried operating systems will process the query correctly and produce an accurate reply?

```
[root@aik /root]# ping -s 1500 x.x.x.x
PING x.x.x.x (x.x.x.x) from y.y.y.y : 1500(1528) bytes of data.
1508 bytes from x.x.x.x: icmp_seq=0 ttl=241 time=1034.7 ms
1508 bytes from host_address (x.x.x.x): icmp_seq=2 ttl=241 time=1020.0
ms
1508 bytes from host_address (x.x.x.x): icmp_seq=3 ttl=241 time=1090.4
ms
```

```
1508 bytes from host_address (x.x.x.x): icmp_seq=5 ttl=241 time=1060.0
ms

--- x.x.x.x ping statistics ---
8 packets transmitted, 5 packets received, 37% packet loss
round-trip min/avg/max = 1000.2/1041.0/1090.4 ms
[root@aik /root]#
```

As it seems all the probed operating systems I have tested this against behaved correctly processing the query and sending the ICMP echo reply back.

What else can assist us with this kind of query?
The DF (Don't Fragment) bit.

Some operating systems will process the query and set the don't fragment bit on the fragments of the reply like we have outlined in the "*DF Bit Playground*" section. These operating systems will be Sun Solaris, AIX 4.3, Linux 2.4.x, and HP-UX 10.30 & 11.0x.

We can use other methods, which does not generate the kind of noise this method generates. Basically there is no reason for this size of ICMP Echo requests, and it should trigger IDS systems immediately and alert them that something suspicious is happening.

7.5 Other Possible Active Fingerprinting Methods and Techniques Using the ICMP Protocol

I have examined several ideas for future ICMP Active Fingerprinting methods. Since I currenly lack the availability of equipment I cannot further investigate these:

- Time elapsed until we receive an ICMP Fragment Reassembly Time Exceeded error message when we will send one fragment to a targeted IP address.
- The rate in which we will receive the ICMP Error messages. This idea is not new BUT nobody payed attention to the fact that different ICMP error messages may have different rates defined. For example with Linux kernel 2.4.x we can set:
    - ICMP Destination Unreachable error message rate
    - ICMP Parameter Problem error message rate
    - ICMP Time Exceeded error message rate

  An attacker can use this to trigger one of the ICMP error messages listed above and count the number of ICMP error messages received within a given amount of time.
- The rate in which we will receive ICMP query message replies. A malicious computer attacker can probe a targeted network with ICMP Echo requests, for example, and count the number of ICMP query reply messages received within a given amount of time.

8.0 The usage of ICMP in The Passive Operating System Fingerprinting Process

8.1 An introduction to Passive Fingerprinting[64]

Passive Fingerprinting is a technique used to map a targeted network (and networks and hosts communicating with it) using sniffed information (exchanged network traffic) from that network.

Different operating systems use different implementations of the TCP/IP stack. We can identify differences between those TCP/IP stack implementations. Therefore differentiate between the different operating systems using those TCP/IP stack implementations differences.

Based on the sniffed information and those differences we can identify various operating systems and services used on the targeted network. We can try to identify host(s), operating systems, and services used on network(s) and host(s) communicating with our target network.

With the traditional active fingerprinting methods one sends a regular or a malformed packet to a targeted host / range of IP addresses and watch for the response. When the response arrives (or not) he will then compare the result with a database holding known fingerprints, which was built earlier, and identify the operating system in use. With active fingerprinting we relate to the uptime of the targeted system, at that particular moment the targeted machine was up and running.

Passive fingerprinting has some advantages over Active Fingerprinting:

- It is able of detecting systems that have low uptime.
- It has better ability to discover services[65]. One example might be if the services were using non-default ports. Another example might be services, which are triggered to operate.
- A machine used for Passive Fingerprinting (or the information collection) will not be detected easily (unless a tool like `anti-sniff`[66] is being used), while active fingerprinting is usually being noticed.
- With Passive Fingerprinting we are able to discover machines behind packet filtering devices, and have more information than an active fingerprinting will produce in similar circumstances.
- Passive fingerprinting has the ability to act on all TCP/IP layers. For example, it gives a malicious computer attacker the ability to learn about certain applications used, which may be unique to the attacked network's environment. This may give us a clear indication about the operating system(s) they are deployed on.
- From a Network view, Passive Fingerprinting can reveal misconfigurations, and even alternative entry points.
- Passive fingerprinting can identify security mechanisms. A good example might be detection of a proxy server or an Authentica & Authorization mechanism.
- The activity will not cause degradation in the attacked network's performance, while active fingerprinting reduces the targeted network's bandwidth.
- The activity will not result in denial of service condition against a machine or a network device in the targeted network (we have seen some cases in the past where a simple `nmap` scan on a Cisco router will cause a Denial-of-Service).

---

64 -Passive Mapping: An Offensive Use of IDS, by Cortez Giovanni
-Passive Mapping: The importance of Stimuli: by Cortez Giovanni
-Passive Fingerprinting, by Lance Spitzner. http://project.honeynet.org/papers/passive/
-Passive Host Fingerprinting, by Max Vision. http://www.whitehats.com/papers/passive/index.html.

65 One notable example would be Trojans using non-default ports.

66 Anti-sniff from l0pht. More information can be found at http://www.l0pht.com.

Passive fingerprinting has some disadvantages as well:

- Limited address space can be checked, since the method rely on user & network activities, and it does not initiate one. We are totally dependent upon information sent and/or network usage.
- Some applications generate their own packets with the application's specific field values and will not produce the same signature(s) as the operating system itself would.
- Some of the default field values we rely upon can be easily changed through simple operating system configuration options.

The Passive Fingerprinting information can be collected from various locations, not only from inside the targeted network.

The quality of the information will be affected from the location of the sensor. If, for example, the location of the sensor will be inside an internal segment, than the entire network communications between internal hosts (and also the outside world) will be revealed, and analyzed. If the location of the sensor will be just outside a filtering device defending the target network, the information gathered will include the host(s) that are allowed to be accessed from the Internet, and to access the Internet only. Most of the internal machines (and infrastructure) will not be revealed with this scenario.

As one can conclude, a deployment of mass distributed networks of Passive Fingerprinting sensors can introduce a major threat. For example, intelligence agencies can invest in increasing their country's Internet bandwidth and speed, in order to force traffic from other countries through the information collecting country's Internet infrastructure. Using this method, other nations traffic will be routed through that country's Internet infrastructure, allowing the intelligence agencies of that country to passively map systems & machines inside those countries and systems they communicate with in other countries as well. Since an analysis at all levels of TCP/IP can be done on the exchanged communication, and presuming the traffic is not encrypted, a great deal of information can be gleaned from this kind of activity[67].

The usage of Passive Fingerprinting techniques is not limited to offensive use only. One can set up a defensive passive fingerprinting systems in order to find unreported systems and services that are in contrast with the security policy of that organization. Since a full analysis can be done on all TCP/IP layers of the information gathered, the defensive system can also track its internal users usage of the Internet (for example web sites they browse).

Is this sound like a mix-up of sniffers and intrusion detection systems abilities all together with a system built defensively?

Because the information is gathered using sensors we can do with it a lot more than just Passive Fingerprinting. This gives a unique add-on to the abilities of a system built for passive collection of data – a sniffer and set of filters that can do:

- Passive fingerprinting
- Intrusion Detection
- Monitoring of internal Users Internet behavior.
- Monitoring of internal Users Internal communications.
- Monitoring for Security Policy breaches.

[67] Passive Mapping: An Offensive Use of IDS, by Cortez Giovanni; Passive Mapping: The importance of Stimuli: By Cortez Giovanni

And a lot more, just use your imagination.

Passive fingerprinting resembles network intrusion detection systems in the way information is gathered. What are the differences between the two? An IDS system role is to detect attacks whether successful or not against a network it deployed on. Passive Fingerprinting methods will identify operating systems (and other kind of information such as services, special applications etc.). These services can be combined together.

Some intrusion detection systems collect information about the system they defend using active fingerprinting methods. This is done in order to discover which TCP/IP stack implementation is being used (and the operating system using it). These scans can sometimes introduce a denial of service condition against the targeted host(s) and/or network devices. The information gathered help the intrusion detection system to build fragmented packets correctly according to its destination operating system's TCP/IP stack behavior. Instead of using active fingerprinting, a passive fingerprinting approach can be used as well; this eliminates the possibility of a denial of service against the residing machines and networking devices. It also brings another important gain – the intrusion detection system can discover, automatically, new systems added to a protected network (we can compare a list of IP addresses we have discovered using passive fingerprinting to the actual IP address list of the organization).

Some of the advantages of a passively mapping process are immediately seen here. One notable example is systems, which have low uptimes. With traditional active fingerprinting methods, one will not notice the existence of those systems, and when information is collected about an outside system tying to access the particular IP address of the low uptime system the right conclusion will not be made.

Today, the passive fingerprinting methods known to the author do not rely on all the information that will be examined and explained in this section. Most of the passive fingerprinting methods rely upon TCP only techniques mainly related to few fields inside the protocol header such as – the initial IP TTL field value, TOS, Windows size, Maximum Segment Size, IP Identification number, Initial Sequence Numbers, the Don't Fragment flag, Sack OK option, nap option, and windows scaling option.

I hope this section will change the regular approach.

8.2 The Quality of the Information Gathered (Location of the Sensor)

A sniffer that can be deployed at various locations regarding the targeted network infrastructure collects the needed information for the passive fingerprinting process. The sniffer can be located not only inside a network, as stated in the introduction, but outside a targeted network as well. The quality of the information will be affected, directly, from its location.

8.2.1 A Sensor Located Inside an Internal Segment

In our first example a sensor is located inside an Internal segment. The location of the sensor allows maximum information to be gathered. One can assume that some network services are internal only, such as an internal E-Mail server, DNS Servers, File Servers, DHCP, etc. The location of the sensor will allow it to discover these services and servers, since internal systems will query these servers. Every host communicating with the outside world and/or the DMZ will be identified and tagged.

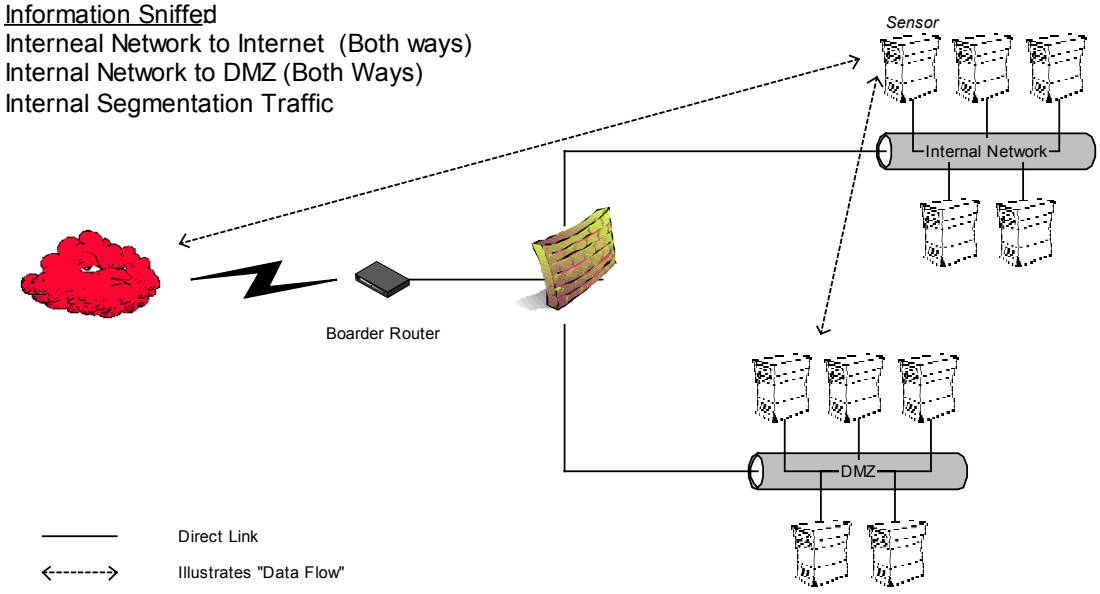


Diagram 9: A Sensor located inside the Internal Network

How a malicious computer attacker will place a sensor inside an internal network? One possibility will be a combination of a virus/Trojan sent attached to an email addressing an internal user. If you will ask yourself how much video clips / pictures / jokes you receive from friends and colleges via email this will sound more real. One example might be a modification of the “LOVE Letter” virus. It will not only send emails to 50 or so people that are listed in your contact info, it will also attach a program, which would do a Passive Fingerprinting information gathering with abilities to send the information gathered back to the malicious computer attacker (can also introduce filters and other gizmos).

8.2.2 A Sensor Located in the DMZ

The next example demonstrates a sensor located in the DMZ. With this example the amount of the information we gather is less than with the prior example. We can only unveil internal systems communicating with the services located in the DMZ as well as outside machines communicating with the various services located in the DMZ[68]. We can also conclude part of a rule base a filtering device, that might be in place, will be using.

A malicious computer attacker can compromise one of the DMZ services, and place his sensor on the compromised machine. We can name few services, which are known to have vulnerabilities in them – some versions of wuftpd, some versions of bind, some versions of IIS and the list is long.

[68] If this is the case, there is a serious misunderstanding in the design of the Network regarding Network Security.

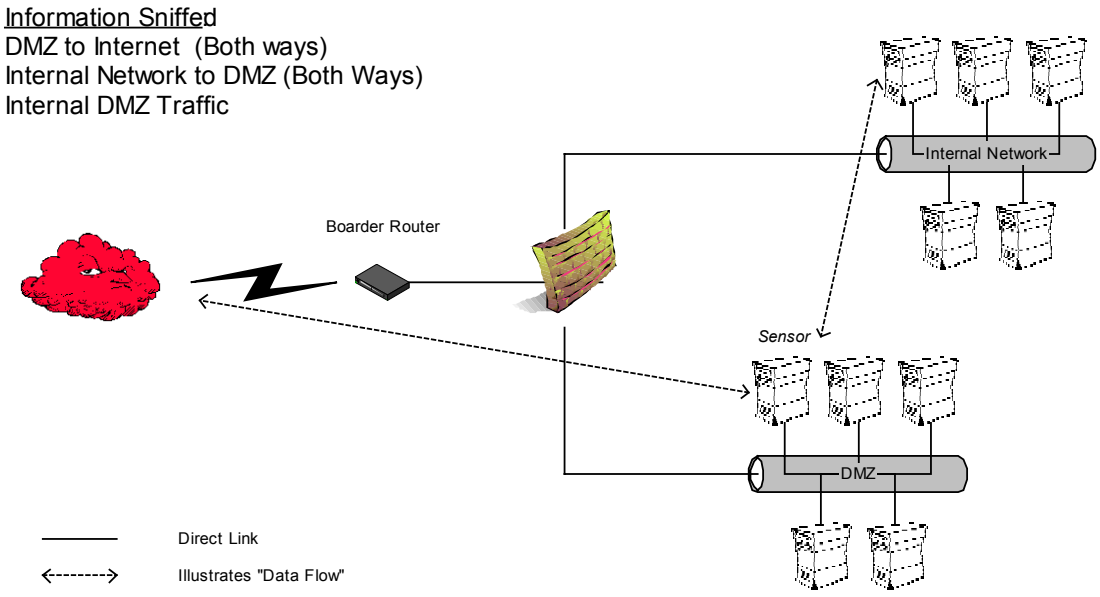


Diagram 10: A Sensor located in the DMZ

8.2.3 A Sensor Located Outside A Targeted Network

Our last example of sensor placement will be the upstream/downstream link of the targeted network and its ISP.

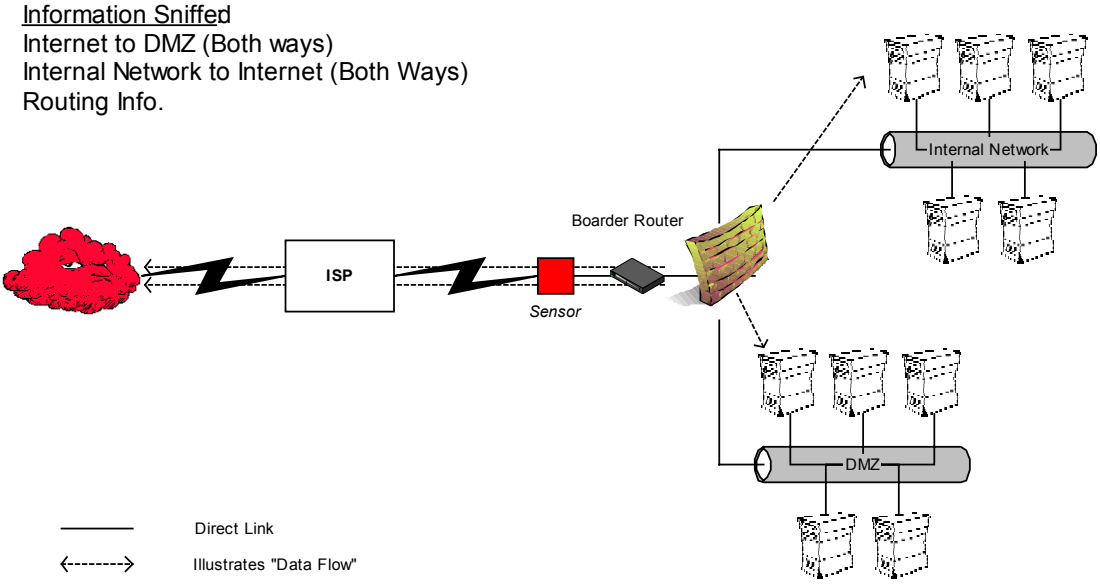


Diagram 11: A Sensor is located on the upstream/downstream link of the attacked network

With this example the information gathered includes traffic flow between internal systems and systems on the Internet, and traffic flow between hosts on the Internet and the DMZ.

Some organizations might have dedicated WAN lines for upstream and downstream links. Whether the sensor is located on the downstream link or on the upstream link it will gather traffic information about “one side of the connection” only.

Sensor located on the upstream link and a sensor located on the downstream link will produce different mapping. This is because the number of Internal systems that access the Internet is usually higher than the number of Internal systems that are allowed to be accessed from the Internet.

Routing information can also be gleaned, if the sensor is to be located between the targeted network’s router(s) to the ISP’s router(s).

8.3 Passive Fingerprinting & ICMP an Introduction

With the ICMP protocol we need to relate to a few different parameters than with other protocols used for Passive Fingerprinting.

The sets of parameters, or questions, we are going to use for the Passive Fingerprinting process with ICMP are:

- Which operating system answers for what kind of ICMP query messages?
- Which operating system answers for special/crafted ICMP queries and how?
- Which operating system produces what sort of ICMP Error messages?
- An analysis of ICMP error messages. Pinpointing several fields inside the IP header and in the ICMP portion of the datagram that will help us identify differences between different operating systems.
- Analyses of ICMP query messages (request & response). Pinpointing several fields inside the IP header and in the ICMP portion of the datagram that will help us to identify differences between operating systems.

The amount of information we can work with is significant, and will allow us to be very accurate in concluding what kind of operating systems produced the ICMP traffic examined, or answered it.

Even if the local administrator has changed few default settings regarding the default behavior of the operating system with certain fields inside the ICMP datagram (IP Header or the ICMP portion of the datagram), other fields will come to our rescue.

In this section I am going to walk through each of the identification stages and demonstrate to you, the reader, how passive fingerprinting with ICMP is done.

8.3.1 Which operating system answers for what kind of ICMP Query messages?

We can divide this section to a number of available probes and answers gathered:

- Regular ICMP query message type traffic
- Advanced ICMP query methods

8.3.1.1 Regular ICMP Query message types traffic

The question “Which operating system answer for what kind of ICMP Query messages?“ help us identify certain groups of operating systems.

The answers gathered would allow us to group together certain operating systems that would answer for a particular normal ICMP Query message(s).

For example, Linux and *BSD based operating systems with default out-of-the-box installation answer for ICMP Echo requests and for ICMP Timestamp Requests. Until Microsoft Windows 2000 family of operating systems has been released it was a unique combination for these two groups of operating systems. Since the Microsoft Windows 2000 operating system family mimics the same behavior (yes mimic), it is no longer feasible to make this particular distinction.

Microsoft might have been thinking that this way of behavior might hide Microsoft windows 2000 machines in the haze. As we have seen they have much more to learn.

The thing is there is no clear distinction between one operating system to another based on this method. We can only group operating systems together and try other methodologies in order to divide those groups a bit more.

Combining the Information

We can correlate the information gathered, and try to conclude which are the involved operating systems.

An example: If we see a certain operating system answer for an ICMP Information request and than answer for ICMP Address Mask request we can conclude that this operating system is an ULTIX based machine.

Producing other interesting data with the ICMP Query replies

We can also introduce another question “How the answering operating systems answers our ICMP queries? Are they producing other interesting data for us?”

Lets look at the next trace:

```
17:10:19.538020 if 4  > y.y.y.y > x.x.x.x : icmp: echo request (ttl
255, id 13170)
                  4500 0024 3372 0000 ff01 9602 yyyy yyyy
                  xxxx xxxx 0800 54a4 8d04 0000 cbe7 bc39
                  8635 0800
17:10:19.905254 if 4  < x.x.x.x > y.y.y.y : icmp: echo reply (DF) (ttl
233, id 24941)
                  4500 0024 616d 4000 e901 3e07 xxxx xxxx
                  yyyy yyyy 0000 5ca4 8d04 0000 cbe7 bc39
                  8635 0800
```

With the example above we have one operating system query another with an ICMP Echo request. The targeted operating system answered the query and set the DF bit with its answer. The fact the DF bit is set with the ICMP Echo reply will limit the number of operating system the replying side might be to: Sun Solaris, HPUX 10.30 & 11.0x., AIX 4.3.x, and Linux Kernel 2.4.x.

Queries aimed at the broadcast/network address

Another piece of information we might use is "Which operating system answer for ICMP queries aimed at the broadcast / network address of the network they reside on?"

For example, Microsoft, and *BSD based operating systems will not answer ICMP query messages aimed at the broadcast address of the network they reside on. A SUN Solaris based operating system machine will answer for ICMP Echo requests and for ICMP Timestamps requests aimed at the broadcast address of the network it resides on.

Common to All

An interesting detail that one should be familiar with is that the only ICMP query message type, which is implemented with all operating systems, is the ICMP Echo request. RFC 1122[69] states that every host should implement an end-user-accessible application interface for sending ICMP Echo request query messages to other hosts. We can test this when we use the "`ping`" utility on various operating systems. "`ping`" uses its own default values for several field values within the ICMP Echo request datagram, and not the operating system's.

So, what will happen if we will see an ICMP query message type coming from a host to another, which is not an ICMP Echo query message?

We can than conclude that the querying host is using a 3rd party utility (not an OS integrated). Than the question will be – *what tools are able of sending this kind of ICMP query message?* And *which kind of operating system can provide the ground for such a tool*? UNIX and UNIX-like operating systems are known to as a better ground for hackery tools[70].

Since we have logged the ICMP datagram sent, we can compare it to a signature produced from different tools and try to locate the tool being used (sometimes it might be needed to compile the tools on different architectures of the same operating system)[71]. Even if the malicious computer attacker was using different parameters each time he used the tool, still the tool might have unique fingerprints. After identifying the tool we might learn about the underlying operating system this tool might have been compiled on, eliminating, our operating system choise selection.

With Passive fingerprinting we must remember that we are only observers and are dependent upon usage of the network.

Countermeasure

Fooling this method of passive fingerprinting is very simple. Just configure your operating system not to answer ICMP query messages. You can configure your operating system not to answer for an ICMP query message it was configured out-of-the-box to answer for. This will fool the database I have described in this section.

You can also change some parameters that will affect the ICMP query request (this is dependent upon configuration options with your operating system).

---

[69] RFC 1122: Requirements for Internet Hosts - Communication Layers, http://www.ietf.org/rfc/rfc1122.txt.

[70] Altough there is a trend to port a lot of *nix based tools to the win32 platform.

[71] See my article "Identifying ICMP Hackery Tools Used In The Wild Today", http://www.sys-security.com/archive/securityfocus/icmptools.html.

8.3.1.2 Advanced ICMP Query Methods

What if some advanced ICMP query methods, like I have introduced in the sections before, will be used in the wild? Than it will allow us, even more accurately, to find the operating system being targeted by another party.

What will happen if a certain field inside the ICMP query message will be sent with a mangled field value? Which operating system will answer the request and do the response will reveal unique information regarding the replying operating system?

8.3.1.2.1 Advanced Host Detection with ICMP[72]

We will concentrate in the ability to trigger several types of ICMP error messages back from a targeted IP address (host).

We will force the target to generate an ICMP error message by mangling a certain field value in our query. We have several field values that we can choose from in order to generate several different ICMP error messages.

All conditions forced by the query host on the targeted IP address, will force the underlying OS kernel to issue an ICMP error message. With only one exception, all the error conditions will always trigger an ICMP error message.

Some of the methods can be abused using a certain tool only.

For example, if we will use a value, which does not represent a valid protocol number field value with the IP header, the targeted host will elicit an ICMP Destination Unreachable Protocol Unreachable error message back to the offending packets IP source address.

`nmap` 2.54 Beta 1 has integrated this and Fyodor has named it - IP Protocol scan. `nmap` sends raw IP packets *without any further protocol header* (no payload) to each specified protocol on the target machine. If an ICMP Protocol Unreachable error message is received, the protocol is not in use. Otherwise it is assumed it is opened (or a filtering device is dropping our packets).

In the next example I have used `nmap` 2.54 beta 22 in order to scan a Microsoft Windows 2000 SP1 Professional machine:

```
[root@godfather /root]# nmap -vv -sO 172.18.2.200
Starting nmap V. 2.54BETA22 ( www.insecure.org/nmap/ )
Host hostname (172.18.2.200) appears to be up ... good.
Initiating IPProto Scan against hostname (172.18.2.200)
The IPProto Scan took 4 seconds to scan 254 ports.
Interesting protocols on hostname (172.18.2.200):
(The 249 protocols scanned but not shown below are in state: closed)
Protocol   State       Name
1          open        icmp
2          open        igmp
6          open        tcp
17         open        udp
47         open        gre


Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

[72] You can find more information in the “Advanced Host Detection methods with the ICMP protocol” section.

A `snort` trace of some of the communication exchanged:

```
05/20/01-13:09:24.502761 172.18.2.201 -> 172.18.2.200
PROTO176 TTL:47 TOS:0x0 ID:8652 IpLen:20 DgmLen:20


05/20/01-13:09:24.502761 172.18.2.200 -> 172.18.2.201
ICMP TTL:128 TOS:0x0 ID:15672 IpLen:20 DgmLen:56
Type:3  Code:2  DESTINATION UNREACHABLE: PROTOCOL UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
172.18.2.201 -> 172.18.2.200
PROTO176 TTL:47 TOS:0x0 ID:8652 IpLen:20 DgmLen:20
Protocol: 0xB0 (unknown or header truncated)** END OF DUMP
00 00 00 00 45 00 00 14 21 CC 00 00 2F B0 0B B9  ....E...!.../...
AC 12 02 C9 AC 12 02 C8 02 C8 02 C8 02 C8 02 C8  ................
```

The method used by `nmap` is easily detected, since the datagrams we will see in our logs will not have payloads. It should turn in `nmap` quite easily.

A malicious computer attacker may use only one datagram with this method. Again, the nature of the datagram, not having a payload, will educate us about the tool used (and the platform it might be compiled on).

The usage of advanced host detection methods will help us map hosts and networking devices onthe network we are targeting, and host(s) communicating/probing it.

8.3.1.2.2 Operating System fingerprinting methods with ICMP (Crafted)

Some new active operating system fingerprinting methods I have found in my research about ICMP can be used as well. For example:

- Using the TOS field inside the IP Header with a method called “TOS Echoing”.
- Using the ‘Unused bit’ inside the IP Header. Sun Solaris and HPUX 11.x will echo back this field when it will be set.
- What will happen if the DF bit will be set with the ICMP requests? Than according to the type of the request sent we can identify several operating systems. This method is called “DF bit echoing”.
- Sending ICMP query message types with the Code field set to a value different than zero.
- What will happen if the ICMP Address Mask request will be fragmented? It will allow us to fingerprint Sun Solaris and HP-UX 11.0 (and probably 10.30) based machines.

With the examples above, as well as with the regular ICMP query messages, if a tool is known to produce this kind of tests (one such tool is `sing`, available from

http://www.sourceforge.net/projects/sing) we might be able to identify it, and also state on which operating systems this tool is known to be compiled on (LINUX, *BSD, Sun Solaris).

One notable distinction between the regular ICMP queries to the advanced ICMP queries is that with some of the advanced methods we can reveal a certain operating system, where with the regular ICMP queries we can only group certain operating systems together or hope that other variants of ICMP query message types will be used against the system in question so we will have more information in hand, that may help us to conclude the operating system in use.

8.3.1.3 How this should work?

Building a database holding the information about which operating system will answer for what ICMP query message type (whether a regular one or an advanced) and than comparing the information we have from the real world to the database.

The database will hold information about operating systems, which will produce other interesting data with the ICMP query replies as well.

An example:
Our sensor have picked the following traffic exchanged:

```
17:23:46.605297 if 4  > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x8]
(ttl 255, id 13170)
                  4508 0024 3372 0000 ff01 60e4 xxxx xxxx
                  yyyy yyyy 0800 0e9a d604 0600 f2ea bc39
                  553c 0900
17:23:46.895255 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x8]
(ttl 243, id 58832)
                  4508 0024 e5d0 0000 f301 ba85 yyyy yyyy
                  xxxx xxxx 0000 169a d604 0600 f2ea bc39
                  553c 0900
```

What can we learn from this sniffed traffic?

- The IP address y.y.y.y is a valid IP address for a machine used in the network we are monitoring.
- Whether the querying IP address is internal or external to the network being probed.
- An ICMP Echo request was sent with a TOS byte field value set to 0x8 hex. The operating system, which has answered the query (y.y.y.y), echoed back the TOS field.
- We can conclude that the probed operating system *is not* Microsoft Windows 2000, Novell Netware or ULTIX. This is based on the TOS field echoing fingerprinting method I have introduced in my research about ICMP.

We have other relevant information we can use here; we will explain those issues later on.

8.3.2 Passive fingerprinting methods using ICMP Error Messages

Which fields or descriptions will interest us the most?

- Operating system, which do not generate ICMP Protocol Unreachable Error Messages
- ICMP Error Message Quenching
- ICMP Error Message Quoting Size
- LINUX ICMP Error Message Quoting Size Differences / The 20 Bytes from No Where
- Foundry Networks Networking Devices Padded Bytes with ICMP Port Unreachable(s) / The 12 Bytes from No Where
- ICMP Error Message Echoing Integrity (Tested with ICMP Port Unreachable)
- Novell Netware Echoing Integrity Bug with ICMP Fragment Reassembly Time Exceeded
- The Precedence bits with ICMP Error Messages (Identifying LINUX)
- TOS Bits (=field) Echoing with ICMP Error
- DF Bit Echoing with ICMP Error Messages

See Section 7.3 for more information.

8.3.3 Analysis of ICMP Query messages (request & reply)

The only ICMP query message type, which is implemented with all operating systems, is the ICMP Echo request. RFC 1122 states that every host should implement an end-user-accessible application interface for sending ICMP Echo request query messages to other hosts. The "`ping`" utility is using this implementation on various operating systems.

Since not all ICMP query request message types are implemented on the various operating systems it leaves us only with ICMP Echo requests to be examined closely.

Please note: "`ping`" uses its own default values for several field values within the ICMP Echo request datagram it generates, and not the Operating System's.

Which information and field values will interested us the most in an ICMP Echo request generated by a "`ping`" utility?

The IP Portion

- The TOS Byte (Precedence Bits, TOS Bits, Unused Bit)
- IP Identification
- The DF Bit
- The Unused Bit

- IP TTL
- IP Options

The ICMP Portion

- ICMP Identification Number
- ICMP Sequence Number
- ICMP Data field (payload)
    - Offset from ICMP Header
    - Content
    - Size
- ICMP Echo Request Total Size

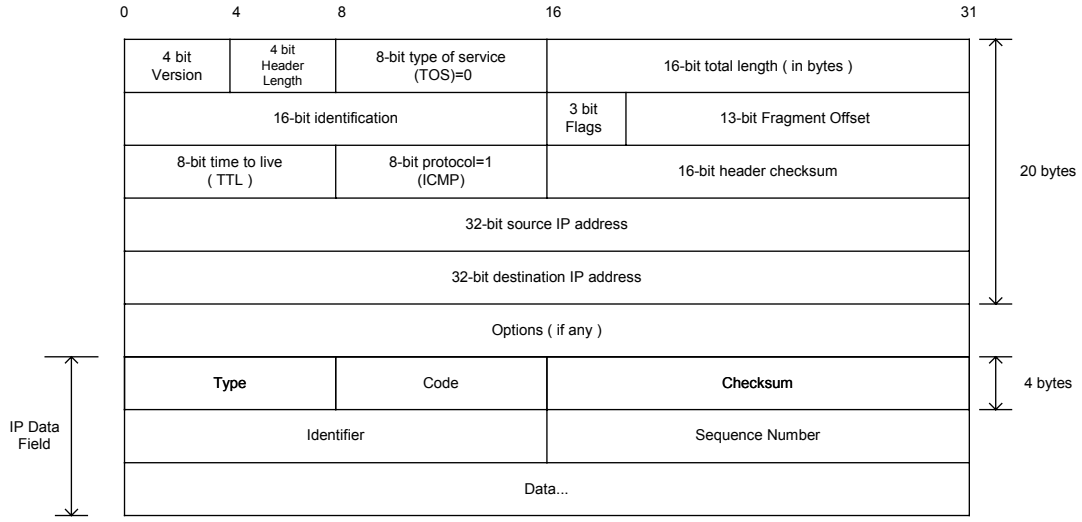


Figure 29: ICMP EHCO Request Message Format

8.3.3.1 The IP Portion

8.3.3.1.1 The TOS Bit

Discussed in Section 7: "Playing With The TOS Field".

8.3.3.1.2 IP Identification field value

Discussed in Section 7: "Fun with IP Identification Field Values"

8.3.3.1.3 The DF Bit

Discussed in Sections 7

8.3.3.1.3.1 DF Bit Echoing
Some operating systems, when receiving an ICMP query message with the DF bit set, will set the DF bit with their replies as well. Sometimes it will be in contrast with their regular behavior, which will be not setting the DF Bit in their replies for a regular query that comes with the DF bit not set.

Discussed in Section 7.

8.3.3.1.4 IP Time-to-Live field value with ICMP
Discussed in Section 7.

8.3.3.1.4.1 IP TTL Values in ICMP Echo Requests
Discussed in Section 7.

8.3.3.1.4.2 IP TTL Values in ICMP Echo Replies
Discussed in Section 7.

8.3.3.1.4.3 Correlating the Information
Discussed in Section 7.

8.3.3.1.5 IP Options
T.B.D.

8.3.3.2 The ICMP Portion
8.3.3.2.1 ICMP Identification Number
RFC 792 – "The Internet Control Message Protocol"[73] define the use of the ICMP Identifier field. It states that the identifier should aid in matching echo requests and replies sent to different machines. The exact wording is "For example, the identifier might be used like a port in TCP or UDP to identify a session".

The parameters we look after are:

- The source of the ICMP ID number
- Initial ICMP ID field value
- Is the same ICMP ID value is assigned to the same host each time we issue the `ping` command?
- The gap between one ICMP ID value to another

[73] RFC 792, The Internet Control Message Protocol. http://www.ietf.org/rfc/rfc0792.txt.

All values represented here were checked after boot.

Operating System	ICMP ID Field Value Starts with HEX / Decimal	Source of the ICMP ID Number	Carry the same ID number to the same host with another ICMP Echo request?	Gap
Linux Kernel 2.2.x Linux Kernel 2.4	13315*	PID (Process ID assigned to Ping)	No	According to other processes in the System
FreeBSD 4.1	57600*	PID (Process ID assigned to Ping)	No	According to other processes in the System
FreeBSD 3.4		PID (Process ID assigned to Ping)	No	According to other processes in the System
OpenBSD 2.7		PID (Process ID assigned to Ping)	No	According to other processes in the System
OpenBSD 2.6		PID (Process ID assigned to Ping)	No	According to other processes in the System
NetBSD		PID (Process ID assigned to Ping)	No	According to other processes in the System
BSDI BSD/OS 4.0		PID (Process ID assigned to Ping)	No	According to other processes in the System
BSDI BSD/OS 3.1		PID (Process ID assigned to Ping)	No	According to other processes in the System
Aix 4.1	11532*	PID (Process ID assigned to Ping)	No	According to other processes in the System
Solaris 2.5.1		PID (Process ID assigned to Ping)	No	According to other processes in the System
Solaris 2.6	2080*	PID (Process ID assigned to Ping)	No	According to other processes in the System
Solaris 2.7		PID (Process ID assigned to Ping)	No	According to other processes in the System
Solaris 2.8		PID (Process ID assigned to Ping)	No	According to other processes in the System
Windows 95 Windows 98 Windows 98 SE	200 / 512		Yes	Value Always = 512 Equals the number first assigned
Windows ME	300 / 768		Yes	Value Always = 768 Equals the number first assigned
Windows NT 4 Workstation SP3 Windows NT 4 Workstation SP6a	100 / 256		Yes	Value Always = 256 Equals the number first assigned
Windows NT 4 Server SP4	100 / 256		Yes	Value Always = 256 Equals the number first assigned

Operating System	ICMP ID Field Value Starts with HEX / Decimal	Source of the ICMP ID Number	Carry the same ID number to the same host with another ICMP Echo request?	Gap
Windows 2000 Family	200 / 512		Yes	Value Always = 512 Equals the number first assigned
Windows 2000 Family with SP1	300 / 768		Yes	Value Always = 768 Equals the number first assigned
* Non-Constant Value				

Table 25: ICMP ID information

8.3.3.2.1.1 The source of the ICMP ID number

The "`ping`" utility with UNIX and UNIX like operating systems will use the Process ID (PID) as its ICMP ID value. Therefore the ICMP ID will change each time we will issue the command.

With the Microsoft based operating systems constant values are used.

This affects the initial ICMP ID field value.

8.3.3.2.1.2 Initial ICMP ID field value

Microsoft Windows operating systems use a constant value for their initial ICMP ID field value. Microsoft Windows NT machines use 256 as their initial ICMP ID filed value (and the older Microsoft based operating systems). Microsoft Windows 98/ 98 SE / Windows 2000 family use 512 as their initial ICMP ID field value. Microsoft Windows ME use 768 as its initial ICMP ID field number – which have made it unique among all Microsoft based operating systems.

With the introduction of Service Pack 1 for Microsoft Windows 2000 family of operating systems the ICMP ID field value has changed from the value of 512 to 768.

The "`ping`" utility with UNIX and UNIX-like operating systems will use the Process ID (PID) as its ICMP ID value.

When examining an ICMP Echo request produced with the "`ping`" utility and identify the values of 256, 512, & 768 as being used for the ICMP ID, we will than conclude that the issuing host is running one of the Microsoft Windows operating systems.

8.3.3.2.1.3 Is the same ICMP ID value is assigned to the same host each time?

We send an ICMP Echo request to a host and receive a reply. We than issue another ICMP Echo request to the same host. Will the ICMP ID field number be the same as it was with the previous request?

With the Microsoft Windows operating systems the answer is yes.

The “ping” utility with UNIX and UNIX-like operating systems will use the Process ID (PID) as its ICMP ID value. Therefore the ICMP ID field value will change each time we issue the command.

In the next example I have sent an ICMP Echo request to a Microsoft Windows 2000 based machine from a LINUX based machine running Kernel 2.2.14:

```
11/01-23:09:51.398772 x.x.x.x -> y.y.y.y
ICMP TTL:64 TOS:0x0 ID:38
ID:52235   Seq:0  ECHO
9F 86 00 3A 85 15 06 00 08 09 0A 0B 0C 0D 0E 0F  ...:............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

11/01-23:09:51.398819 y.y.y.y -> x.x.x.x
ICMP TTL:255 TOS:0x0 ID:39
ID:52235   Seq:0  ECHO REPLY
9F 86 00 3A 85 15 06 00 08 09 0A 0B 0C 0D 0E 0F  ...:............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567


god:~# ps aux | grep ping
mia       3020  0.5  0.8  1360  540 pts/0    S    17:14   0:00 ping -c
5 192.168.1.5
root      3022  0.0  0.6  1108  400 pts/2    S    17:14   0:00 grep
ping
god:~#
```

2030 in hex is 52235 decimal.

8.3.3.2.1.4 The gap between one ICMP ID value to another
With UNIX and UNIX-like based operating systems this will be dependent upon other processes in the system.

Since Microsoft based operating systems use constant values for the ICMP ID field value, there is no gap between one ICMP ID number to another.

8.3.3.2.1.5 The usage of ICMP ID and Sequence Numbers with Microsoft Based Operating Systems
RFC 792 (Internet Control Message Protocol) suggests how the ICMP Identifier field and the ICMP Sequence Number field should be used:

“The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply”.

It literally suggests that the ICMP Identifier field will be used to differentiate between ICMP Query messages sent to different hosts. It also suggests that the ICMP Sequence Number field will be used to differentiate between the ICMP query messages sent to the same host.

The '`ping`' utility with UNIX and UNIX-like operating systems has adopted this suggestion.

When examining the behavior of the '`ping`' utility with Microsoft Windows based operating systems I have encountered a different behavioral pattern.

The next example is a trace (using the `windump` program - http://netgroup-serv.polito.it/windump/install/Default.htm) of ICMP Echo requests initiated by lunching two ping commands at the same time from a Microsoft Windows 2000 SP1 operating system based machine. One instant was aimed at the host 172.18.1.2, and the other at the host 172.18.1.134:

```
E:\>windump -xnvv -s 1600 icmp
windump: listening on\Device\Packet_{79C233F1-6CD7-49EB-8FA2-
FA825CB1C9C3}
11:31:21.848025 172.18.1.179 > 172.18.1.2: icmp: echo request (ttl 128,
id 11071)
                         4500 003c 2b3f 0000 8001 b4a8 ac12 01b3
                         ac12 0102 0800 265c 0300 2400 6162 6364
                         6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                         7576 7761 6263 6465 6667 6869

11:31:22.221772 172.18.1.179 > 172.18.1.134: icmp: echo request (ttl
128, id 11075)
                         4500 003c 2b43 0000 8001 b420 ac12 01b3
                         ac12 0186 0800 255c 0300 2500 6162 6364
                         6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                         7576 7761 6263 6465 6667 6869

11:31:22.844726 172.18.1.179 > 172.18.1.2: icmp: echo request (ttl 128,
id 11077)
                         4500 003c 2b45 0000 8001 b4a2 ac12 01b3
                         ac12 0102 0800 245c 0300 2600 6162 6364
                         6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                         7576 7761 6263 6465 6667 6869

11:31:23.215222 172.18.1.179 > 172.18.1.134: icmp: echo request (ttl
128, id 11078)
                         4500 003c 2b46 0000 8001 b41d ac12 01b3
                         ac12 0186 0800 235c 0300 2700 6162 6364
                         6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                         7576 7761 6263 6465 6667 6869

11:31:23.846116 172.18.1.179 > 172.18.1.2: icmp: echo request (ttl 128,
id 11079)
                         4500 003c 2b47 0000 8001 b4a0 ac12 01b3
                         ac12 0102 0800 225c 0300 2800 6162 6364
                         6566 6768 696a 6b6c 6d6e 6f70 7172 7374
                         7576 7761 6263 6465 6667 6869

11:31:24.216645 172.18.1.179 > 172.18.1.134: icmp: echo request (ttl
128, id 11080)
```

```
4500 003c 2b48 0000 8001 b41b ac12 01b3
ac12 0186 0800 215c 0300 2900 6162 6364
6566 6768 696a 6b6c 6d6e 6f70 7172 7374
7576 7761 6263 6465 6667 6869
```

As it can be seen, the ICMP Identifier field value is the same with both instances. This is regardless the fact we are using the '`ping`' utility to send ICMP Echo requests to two separate hosts. The number assigned to this field is 768 decimal.

So how does the '`ping`' utility with Microsoft based operating systems differentiate between the different ICMP Queries?

The '`ping`' utility is using the Sequence Number field. For each ICMP Echo Request the ICMP Sequence Number is a unique number. The gap between one ICMP Sequence Number field value to another is 100 hex/256 decimal.

This raises another interesting question.

If the ICMP Identifier field has a constant value, can we identify the different Microsoft operating systems passively when someone is using the 'ping' utility to query our machines?

Yes.

Microsoft Windows NT - 256

Microsoft Windows 98/98SE - 512
Microsoft Windows 2000 - 512

Microsoft Windows ME – 768
Microsoft Windows 2000 Family with SP1/SP2 - 768

With the '`ping`' utility with Microsoft based operating systems the values assigned for the different ICMP datagram fields are OS based (in contrast with the '`ping`' utility on UNIX and UNIX-like operating systems which uses the application own values for the different ICMP datagram fields). When using other applications with Microsoft based operating systems to generate ICMP Query messages the ICMP Identifier field values will still be the same as it was with the '`ping`' utility, if these applications will be using the Microsoft MFC.

Therefore when ever we see an ICMP Query datagram with an ICMP Identifier field value of 256/512/768 it will indicate that the underlying operating system to be used is an MS based.

We can also look at the ICMP Sequence Number field value for extra information. The '`ping`' utility with MS based operating systems will issue its first ICMP Query message with the ICMP Sequence Number field set to a value of 256 (the '`ping`' utility with UNIX and UNIX-like operating systems will have this field value set to 0 on its first query to a Host). This field value will increase with 256 decimal each time we send an ICMP Query message (with the UNIX and UNIX-like '`ping`' utility the field value will increase only if we are sending sequential Queries. Each time we issue the 'ping' command this field value will be set to 0 on the first query to be sent).

We can even calculate the number of ICMP Query messages a Windows based OS have issued since the last boot time. All we need to do is divide the ICMP Sequence number field value with 256.

Microsoft can argue that their ICMP implementation is not in contrast with RFC 792, since the term that was used in order to describe the usage of the ICMP Identifier field was “may be used”. But if we use common sense, than what role, in the Microsoft case, the ICMP Identifier field has?

8.3.3.2.2 Sequence Number

RFC 792 states that the sequence number might be increased for each ICMP echo request sent, and used to distinguish between one ICMP Echo request to another sent to the same host.

The parameters we look for are:

- Initial value of the Sequence Number
- The Gap between one sequence number to another

Since ICMP Echo requests carry with them the sequence number, and its starting number changes from one implementation to another, along with the fact that the gap between each number is different between one operating system to another it gives us another important piece of information to identify various operating systems on the passively probed network.

	Field Value Starts with	Gap between each sequence number HEX / Decimal
Linux Kernel 2.2.x Linux Kernel 2.4	0	100 / 256
FreeBSD 4.1 FreeBSD 3.4 OpenBSD 2.7 OpenBSD 2.6 NetBSD BSDI BSD/OS 4.0 BSDI BSD/OS 3.1	0	100 / 256
Aix 4.1	0	1 / 1
Solaris 2.5.1 Solaris 2.6 Solaris 2.7 Solaris 2.8	0	1/ 1
Windows 95 Windows 98 Windows 98 SE Windows ME Windows NT 4 Workstation SP3 Windows NT 4 Workstation SP6a Windows NT 4 Server SP4 Windows 2000 Professional SP1 Windows 2000 Server SP1 Windows 2000 Advanced Server	256 256 256 256 256 256 256	100 / 256 100 / 256 100 / 256 100 / 256 100 / 256 100 / 256 100 / 256

Table 26: ICMP Sequence Number information

8.3.3.2.2.1 Start value of the Sequence Number

We can differentiate between one group of operating systems to another based upon the initial value used for the sequence number field.

Microsoft based operating systems will use 256 as their initial ICMP sequence number field value, while UNIX and UNIX-like operating systems will use the value of 0.

8.3.3.2.2.2 The gap between one Sequence number to another

If we logged two ICMP Echo requests coming from the same host one after the other in a series than we can look at the gap between the sequence numbers that were used.

Microsoft based operating systems as well as Linux based machines and *BSD based machines, use a gap of 256 (100 Hex) between one sequence number to another sent in a series to the same host.

AIX and Sun Solaris, as well as other UNIX and UNIX-like operating systems, will use a gap of 1 (1 Hex).

8.3.3.2.2.3 Combining the parameters

Combining the two parameters together will help us to identify the group of Microsoft based operating systems, AIX and Sun Solaris, and Linux based machines.

In the next example I have sent an ICMP Echo requests from an AIX 4.1 machine to a Linux based on Kernel 2.2.14 machine. The sequence number is in bold:

```
23:40:17.271616 ppp0 < host_address > target_host_address: icmp: echo
request (ttl 239, id 2233)
                  4500 0054 08b9 0000 ef01 5403 xxxx xxxx
                  yyyy yyyy 0800 1baf 2d0c 0000 3992 670d
                  000c 2396 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
23:40:17.271652 ppp0 > target_host_address > host_address: icmp: echo
reply (ttl 255, id 494)
                  4500 0054 01ee 0000 ff01 4ace yyyy yyyy
                  xxxx xxxx 0000 23af 2d0c 0000 3992 670d
                  000c 2396 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
23:40:18.261619 ppp0 < host_address > target_host_address: icmp: echo
request (ttl 239, id 2235)
                  4500 0054 08bb 0000 ef01 5401 xxxx xxxx
                  yyyy yyyy 0800 14c4 2d0c 0001 3992 670e
                  000c 2a7f 0809 0a0b 0c0d 0e0f 1011 1213
                  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
                  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
                  3435 3637
23:40:18.261665 ppp0 > target_host_address > host_address: icmp: echo
reply (ttl 255, id 497)
```

```
4500 0054 01f1 0000 ff01 4acb yyyy yyyy
xxxx xxxx 0000 1cc4 2d0c 0001 3992 670e
000c 2a7f 0809 0a0b 0c0d 0e0f 1011 1213
1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
3435 3637
```

The sequence number starts at 0, increased by 1.

8.3.3.2.3 Data Field (Payload)

RFC 792 describes the process of creating an ICMP Echo request: The sending side initializes the identifier (used to identify Echo requests aimed at different destination hosts) and sequence number (if multiple Echo requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP Echo to the destination host. In the ICMP header the code equals zero. The recipient should only change the type to Echo reply and return the datagram to the sender.

The RFC does not specify how much data should be sent along with the ICMP Echo request. It only states, "Adds some data". The RFC does not specify what is the data sent as well.

Those two parameters are the root of this section.

The parameters we relate to with the ICMP data field with ICMP Echo request are:

- The Offset of the data field portion from the ICMP Header
- The size of the data field
- The content of the data field

8.3.3.2.3.1 The Offset of the data portion from the end of the ICMP Header

With Microsoft based operating system's "`ping`" utility the Echo Request's data portion comes right after the end of the ICMP datagram IP Header.

With UNIX and UNIX-like operating systems the first 8 bytes of the ICMP data portion are being used for the calculation of the round trip time (RTT)[74].

8.3.3.2.3.2 The size of the data field

Operating System	Size of ICMP Data Field	Total Datagram Size
LINUX Kernel 2.0.x, 2.2.x, 2.4.x	56	84
FreeBSD 4.x	56	84
AIX 4.x	56	84
Sun Solaris 2.x	56	84

[74] This brings another creative thought into mind – How does Microsoft based operating systems calculate the round trip time. It is quite obvious that it somehow "remembered".

Operating System	Size of ICMP Data Field	Total Datagram Size
Microsoft Windows 98se	32	60
Microsoft Windows ME	32	60
Microsoft Windows NT sp6a	32	60
Microsoft Windows 2000 Family	32	60

Table 27: Different ICMP data field size(s) and Total Datagram size(s)

The `ping` utility with UNIX and UNIX-like operating systems use 56 bytes for the ICMP data field (8 bytes of those are used for the calculation of the round trip time).

Microsoft Windows operating systems use 32 bytes for the ICMP data field portion.

If HPUX 10.30/11.0x or AIX 4.3.x use the path MTU discovery process that is based on ICMP Echo requests, we will see ICMP Echo request datagrams with size bigger than 576 bytes. An example was given in the text for a datagram with the size of 1500 bytes.

8.3.3.2.3.3 The content of the data field

Within the ICMP Echo request data field we can find differences between the different operating systems as well.

The `ping` utility with Microsoft Windows based operating system will use the combination of "`abcdefghijklmnopqrstuvwabcdefghi`" in its ICMP Data field.

While the ping utility with UNIX and UNIX-Like operating systems will use "`08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37`" which its ASCII equivalent is "`.....!"#$%&'()*+,-./01234567`".

Again, the `ping` utility with UNIX and UNIX-Like operating systems use its first 8 bytes of the ICMP Data field for calculating the round trip time, and only than put the arbitrary code.

8.3.3.2.3.4 Examples of the ICMP Data Portion

UNIX and UNIX-Like Operating Systems

LINUX based on 2.2.14 kernel

```
08/08-11:57:58.562434 10.0.0.105 -> 10.0.0.103
ICMP TTL:64 TOS:0x0 ID:246
ID:13315   Seq:0  ECHO
96 CB 8F 39 94 93 08 00 08 09 0A 0B 0C 0D 0E 0F  ...9............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

08/08-11:57:59.557629 10.0.0.105 -> 10.0.0.103
ICMP TTL:64 TOS:0x0 ID:250
ID:13315   Seq:256  ECHO
97 CB 8F 39 25 82 08 00 08 09 0A 0B 0C 0D 0E 0F  ...9%...........
```

```
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

FreeBSD 4.1 ICMP Echo Request

```
08/08-22:17:42.264667 192.168.1.15 -> 192.168.1.200
ICMP TTL:255 TOS:0x0 ID:16
ID:57600   Seq:0  ECHO
52 A6 24 6E 41 CC 0E 00 08 09 0A 0B 0C 0D 0E 0F  R.$nA...........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

Aix 4.1

```
08/10-23:41:11.991616 18.170.1.172 -> 139.92.207.58
ICMP TTL:239 TOS:0x0 ID:2271
ID:11534   Seq:0  ECHO
39 92 67 44 00 0B 89 1F 08 09 0A 0B 0C 0D 0E 0F  9.gD............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

08/10-23:41:14.141619 18.170.1.172 -> 139.92.207.58
ICMP TTL:239 TOS:0x0 ID:2275
ID:11534   Seq:2  ECHO
39 92 67 46 00 0B 95 61 08 09 0A 0B 0C 0D 0E 0F  9.gF...a........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

This is a good example where the second ICMP Echo request did not make it to the destination system.

A series of ICMP Echo Requests performed by Sun Solaris 2.6:

```
08/10-23:33:51.861619 18.170.2.161 -> 139.92.207.58
ICMP TTL:239 TOS:0x0 ID:48690  DF
ID:2097   Seq:0  ECHO
39 93 10 DE 00 0A 62 68 08 09 0A 0B 0C 0D 0E 0F  9.....bh........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567

08/10-23:33:52.661614 18.170.2.161 -> 139.92.207.58
ICMP TTL:239 TOS:0x0 ID:48692  DF
ID:2097   Seq:1  ECHO
39 93 10 DF 00 0A 43 3C 08 09 0A 0B 0C 0D 0E 0F  9.....C<........
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

Microsoft Based Operating Systems

Microsoft Windows ME ICMP Echo request:

```
08/08-12:26:21.428181 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:68
ID:768   Seq:256  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

Microsoft Windows NT 4 Workstation SP6a ICMP Echo request:

```
08/08-12:34:12.062116 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:27904
ID:256   Seq:256  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

Microsoft Windows 2000 Professional ICMP Echo request:

```
08/09-17:45:44.496774 10.0.0.103 -> 10.0.0.105
ICMP TTL:128 TOS:0x0 ID:692
ID:512   Seq:256  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

9.0 Filtering ICMP on your Filtering Device to Prevent Scanning Using ICMP

9.1 Inbound

An example of incoming ICMP traffic that should be blocked in order to *prevent scanning techniques that were outlined in this paper* might be:

- All ICMP Query requests

    - ICMP Echo request
    - ICMP Timestamp request
    - ICMP Address Mask request
    - ICMP Information request

  They all can be used in Host Detection, Invrerse Mapping, and OS fingerprinting attempts.

- ICMP Query replies
  Can be used for Inverse Mapping, and to trigger ICMP erroe messages.

- ICMP Error Messages
  Can be used as a Covert Channel
  .
- Deny access to your Broadcast and Network addresses from the Internet. Configure your Router and your Firewall to do so.

If you examine the list closely all ICMP message types, whether query types or error types are listed.

9.2 Outbound

There are people who claim that any traffic type of ICMP should be allowed from a protected network to the Internet. This is not true, in my opinion. Filtering the incoming traffic does not mean we are protected from some of the security hazards I have outlined in this paper.

- ICMP Echo reply (Type 0)
  Can be used to map a Host. Can also be used as a Covert Channel.

- ICMP Destination Unreachable Error Messages (Type 3 familiy)
  I have demonstrated that host detection can be done with bad IP Header packets, which elicit various ICMP Parameter Problem and ICMP Destination Unreachable error messages from the probed machines and draw the targeted network topology.

- ICMP “Fragmentation Needed but the DF bit was set” (Type 3 Code 4)
  To prevent misconfiguration errors as outlined in the “Advanced Host Detection” section.

- ICMP Echo request (Type 8)
  We have to have a true stateful filtering device that will perform Stateful inspection with ICMP in order to let ICMP Echo requests out, and receive only the corresponding ICMP Echo replies.

The current state with filtering devices is not that good. Most of the products in the market today do not perform true stateful inspection with the ICMP protocol. Allowing ICMP Echo replies inside your protected network is very dangerous and simply not worth the risk.

Unless you use a true stateful filtering device with the ICMP protocol don't let ICMP Echo replies into your protected network. This will make your requests useless. block them as well.

- ICMP Time-To-Live Exceeded in Transit (Type 11 Code 0)
  To eliminate the possibility of using traceroutes and Inverse Mapping techniques against your network block this ICMP error message.

- ICMP Fragment Reassembly Time Exceeded (Type 11 Code 1)
  One of the Advanced Host Detection methods was sending only few fragments of a fragmented datagram, and forcing the targeted host to issue an ICMP Fragment Reassembly Time Exceeded error message back to the offending packet's source IP address (the malicious computer attacker).

- ICMP Parameter Problem
  We have demonstrated that we can use packets with bad IP header field values to elicit various ICMP parameter problem error messages from a targeted machine.

- ICMP Timestamp request & reply
  Can be used with Host Detection and Inverse Mapping techniques.

- ICMP Address Mask request & reply
  Can be used with Host Detection and Inverse Mapping techniques

9.3 The liability Question

System administrators / Network administrators don't want to be held liable for an attack attempt generated from their network by an abusive user (or a malicious computer attacker using a compromised system within the network). Therefore blocking some types of ICMP traffic from the protected network to the outside world is recommended because of liability reasons:

- Destination Unreachable Codes 2-4
  - ICMP Destination Unreachable error messages codes 2-4 ("Port Unreachable", "Protocol Unreachable" and "Fragmentation Needed and DF Flag was Set") are a group of messages that are hard error conditions and when received should terminate a connection.
    - This allow a malicious computer attacker to send *fake* ICMP Destination Unreachable codes 2-4 error messages to terminate valid connections between the attacked target and other hosts on the Internet.
    - Old TCP/IP implementations terminate TCP connections when receiving these error messages. Modern TCP/IP implementations no longer terminate a TCP connection when receiving these error messages
- Source Quench messages
  - Since hosts still react to ICMP Source Quenches by slowing communication, they can be used as a Denial-of-Service measure.

- Redirect messages
    - If you can forge ICMP Redirect error messages, and if your target host pays attention to them - ICMP redirects may be employed for denial of service attacks, where a host is sent a route that loses it connectivity, or is sent an ICMP Network Unreachable packet telling it that it can no longer access a particular network.

This means that all outbound ICMP traffic should be disallowed.

9.4 Other Considerations

If you want to maintain strong ICMP filtering rules with your Firewall/Filtering Device I suggest you block all incoming ICMP traffic except for Type 3 Code 4 ("Fragmentation Needed but the Don't Fragment Bit was Set"), which is being used by the Path MTU Discovery process[75]. ICMP Type 3 Code 4 should be allowed from the Internet to your DMZ at least. Opening your Internal segmentation to this kind of traffic is questionable and depends on the facilities / activities / usage of the site and the level of filtering you wish to maintain.

It is also depends upon the operating systems you are using in your network segments. Some operating systems will react differently to ICMP fragmentation needed but the DF bit was set error messages than other operating systems.

If you will block incoming ICMP "Fragmentation Needed but the Don't Fragment Bit was Set" error messages, your network performance will suffer from degradation. You should understand the security risks involving in opening this kind of traffic to your DMZ (& protected network) - The possibility of a Denial-of-Service, Inverse Mapping, Host Detection, and a one-way Covert communication channel (which was not been seen in the wild yet).

Another consideration can be the usage of network troubleshooting tools such as `traceroute` and `ping`. In the case of `traceroute` if the filtering device you are using does not support true stateful inspection with ICMP than allowing ICMP TTL Exceeded In Transit (Type 11, code 0) error messages inside the protected network can lead to various security hazards. The same goes with `ping`, where ICMP Echo reply is even more dangerous when allowed inside the protected network (Inverse Mapping, Covert Channel and more security risks).

You can limit the number of systems that need to use the network troubleshooting tools with ACL, but bear in mind that these systems, probably, can be mapped from the Internet – and this is only the tip of the iceberg.

Internal Host(s) performance considerations – When blocking incoming ICMP Destination Unreachable Network/Host/Protocol/Port Unreachable ICMP error messages coming from the Internet, host(s) will hang when the destination system's network is unreachable/when a host is unreachable/when a protocol on the destination machine is not available/a port on a destination machine is closed. They all will hang until the timeout counter will reach zero. This little inconveniency is better than having the dangers, other types of ICMP error messages inside your network can introduce.

[75] See Section 2 for more information on the Path MTU Discovery process.

Unless your filtering device is a real intelligent one, doing his work with dynamic tables and correlating correctly the ICMP replies with the ICMP requests, do not open your Internal network segment to ICMP traffic at all.

Some might offer to use a Proxy server with the ICMP protocol as a barrier between the Internet and your protected network(s). A Proxy Server is only a tunnel – remember that.

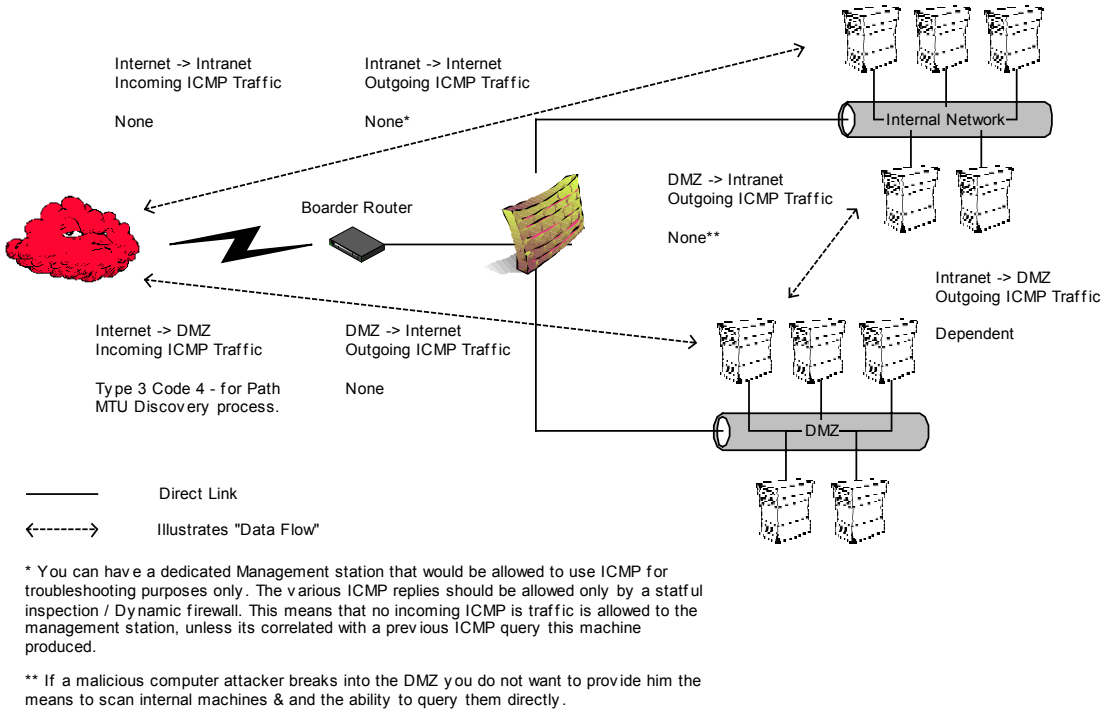


Figure 30: Firewall ICMP Filtering Rules

9.5 Other Problems – Why it is important to filter ICMP traffic in the Internal segmentation

Consider the following realistic scenario:

You have an Internal segment built with Microsoft based operating system machines (for the sake of the example only). A malicious computer attacker might send you a Trojan that will have Host detection and/or mapping capabilities. It will be hidden in an Email message (either as attachment or some other thing) a naïve user will open. After activation it will start to map internal hosts and internal segments and send the information back to the malicious computer attacker.

What will be the easiest method in order to map internal host(s)? Ping them probably.

How many of you reading this research have “management segments” that are allowed to use the `ping` utility in order to verify that some Hosts are alive?

If something like this Trojan gets its way to this segment than probably your entire internal networking infrastructure (or important part of it) will be revealed.

Some one might think that strong filtering or a good anti virus might help – forget it. I have seen people separating their work environment to more than two or three computers, but they always use the Email, and need to surf the web… (good ways to send the collected information out).

My suggestion is to configure internal host(s) not to answer for ICMP query message types they should not answer for. I would restrict this to the maximum and not allow internal hosts to be queried with any ICMP query message type.

If you can afford it, install a personal firewall on all internal hosts, and confiure it to block all ICMP queries.

Back to our monitoring problem - If you need to maintain management/monitoring capabilities, than I would suggest filtering the traffic in both ways from the management stations to the monitored systems in a way it will not be possible to simply query the last (dynamic filtering / stateful filtering with ICMP). Use a dedicated system for the querying and block the other machines in the management segment from doing so.

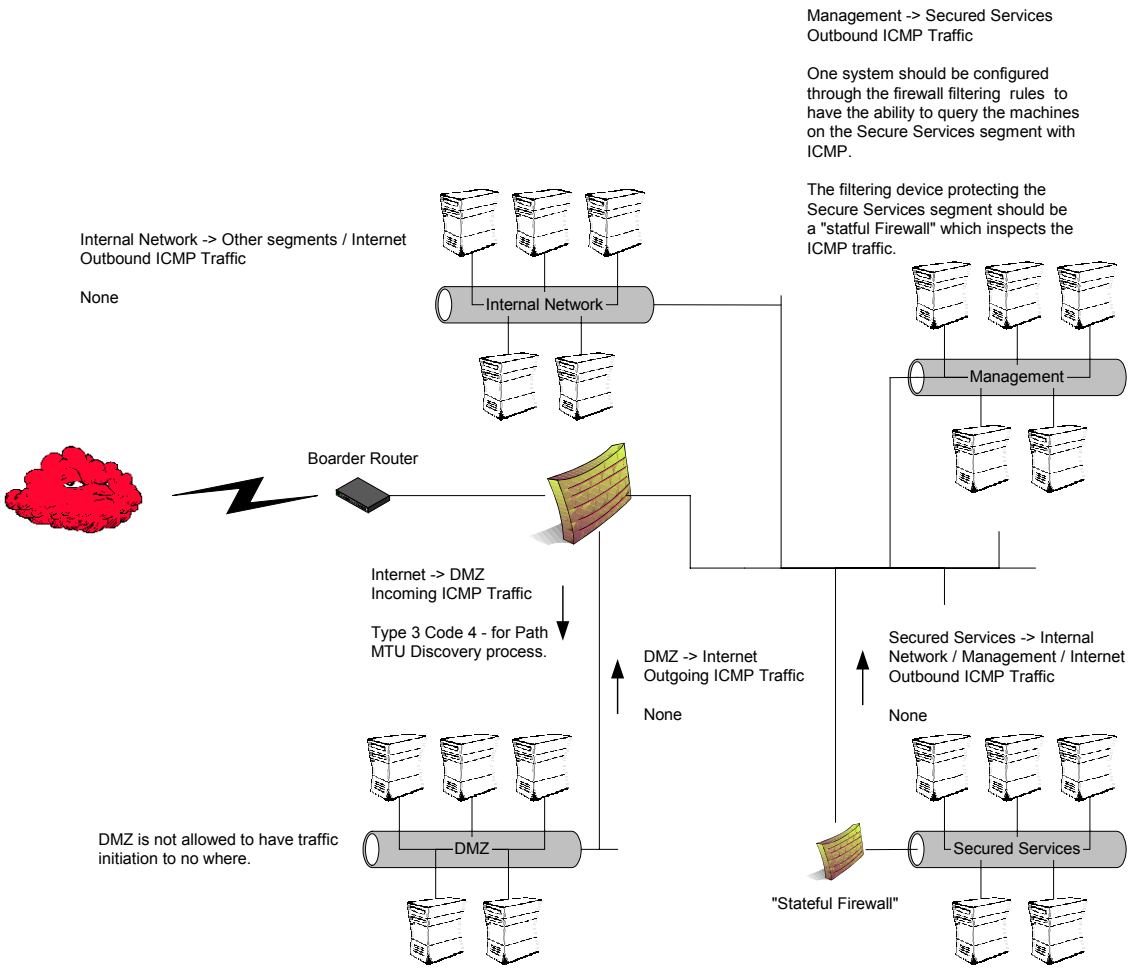


Figure 31: Internal segmentation ICMP Filtering Example

9.6 The Firewall

It is extremely important to block traffic, which is aimed at the Firewall itself. This rule will not block every thing. For example, ICMP error messages the firewall generates for various stimulus.

Some firewalls will hold a certain portion of a fragmented packet until the IP Header and the underlying protocol's header arrives. Sometimes, the ICMP error message for Fragment Reassembly Time Exceeded will not be of the Host, it will be of the Firewall spoofing it. Some Firewalls has the ability to spoof ICMP Echo replies for Hosts they are defending. We will have the opportunity to fingerprint the operating system, which the firewall software is installed on.

We will gain an extremely important ability. Therefore it is recommended to have two basic rules when you configure your firewall's rule base. The first rule will be to deny any traffic destined to the firewall and the second rule will be to deny any error messages (or other conditions such as TCP reject etc.) initiated from the Firewall destined the Internet that might help a malicious computer attacker in his task to fingerprint the Firewall itself.

10.0 Conclusion

The ICMP protocol is a very powerful tool in the hands of smart malicious computer attackers. Mapping, detecting, and fingerprinting of hosts and networking devices can be done in various ways as I have outlined in this paper.

It is extremely important to understand that ICMP traffic can be used for other malicious activities other than scanning, such as:

- Denial of Service Attacks
- Distributed Denial of Service Attacks
- Covert Channel Communications

Therefore filtering Inbound and Outbound ICMP traffic is very important and may help you in preventing risks to your computing environment.

11.0 Acknowledgment

11.1 Acknowledgment for version 1.0

I would like to thank the following people for their help with/during this research.

Ariel Pisetsky for going over this paper correcting my English, and for his moral support.

Christopher Tresco, Systems Administrator at the Massachusetts Institute of Technology provided necessary test systems to verify my findings.

Special thanks to mr2940 for his patience while I introduced my new ideas.

James Cudney, Michael, Pat, for their support when the times where bad.

11.2 Acknowledgment for version 2.0

I would like to thank Alfredo Andres Omella author of `SING` for his help.

I would like to thank Fyodor for his help providing me with necessary test systems.

I would like to thank the people who provided feedback to the first version of this research paper, and to the people who provided feedback to my Bugtraq posts.

11.3 Acknowledgment for version 2.5

I would like to thank Alfredo Andres Omella author of `SING`, for implementing some of the ideas I had into his tool.

I would like to thank Fyodor for his help providing me with necessary test systems.

Christopher Tresco, Systems Administrator at the Massachusetts Institute of Technology provided necessary test systems to verify my findings.

I would like to thank Simple Nomad for his support.

I would like to thank the huge amount of people who provided feedback for my work.

11.4 Acknowledgment for Version 3.0

I would like to thank the following people for their contribution/support:

- JD Glaser
- Jeff Moss
- Lance Spitzner
- Marty Roesch
- Simple Nomad
- Max Vision

I would like also to thank the huge amount of people who provided feedback for my work.

Appendix A: Protocol Numbers

Taken from the IANA list available from: http://www.isi.edu/in-notes/iana/assignments/protocol-numbers.

```
In the Internet Protocol version 4 (IPv4) [RFC791] there is a field,
called "Protocol", to identify the next level protocol.  This is an 8
bit field.  In Internet Protocol version 6 (IPv6) [RFC1883] this field
is called the "Next Header" field.

Assigned Internet Protocol Numbers

Decimal     Keyword      Protocol
-------     -------      --------
     0      HOPOPT       IPv6 Hop-by-Hop Option
     1      ICMP         Internet Control Message
     2      IGMP         Internet Group Management
     3      GGP          Gateway-to-Gateway
     4      IP           IP in IP (encapsulation)
     5      ST           Stream
     6      TCP          Transmission Control
     7      CBT          CBT
     8      EGP          Exterior Gateway Protocol
     9      IGP          any private interior gateway
                         (used by Cisco for their IGRP)
    10      BBN-RCC-MON  BBN RCC Monitoring
    11      NVP-II       Network Voice Protocol
    12      PUP          PUP
    13      ARGUS        ARGUS
    14      EMCON        EMCON
    15      XNET         Cross Net Debugger
    16      CHAOS        Chaos
    17      UDP          User Datagram
    18      MUX          Multiplexing
    19      DCN-MEAS     DCN Measurement Subsystems
    20      HMP          Host Monitoring
    21      PRM          Packet Radio Measurement
    22      XNS-IDP      XEROX NS IDP
    23      TRUNK-1      Trunk-1
    24      TRUNK-2      Trunk-2
    25      LEAF-1       Leaf-1
    26      LEAF-2       Leaf-2
    27      RDP          Reliable Data Protocol
    28      IRTP         Internet Reliable Transaction
    29      ISO-TP4      ISO Transport Protocol Class 4
    30      NETBLT       Bulk Data Transfer Protocol
    31      MFE-NSP      MFE Network Services Protocol
    32      MERIT-INP    MERIT Internodal Protocol
    33      SEP          Sequential Exchange Protocol
    34      3PC          Third Party Connect Protocol
    35      IDPR         Inter-Domain Policy Routing Protocol
    36      XTP          XTP
    37      DDP          Datagram Delivery Protocol
    38      IDPR-CMTP    IDPR Control Message Transport Proto
    39      TP++         TP++ Transport Protocol
```

Decimal	Keyword	Protocol
40	IL	IL Transport Protocol
41	IPv6	Ipv6
42	SDRP	Source Demand Routing Protocol
43	IPv6-Route	Routing Header for IPv6
44	IPv6-Frag	Fragment Header for IPv6
45	IDRP	Inter-Domain Routing Protocol
46	RSVP	Reservation Protocol
47	GRE	General Routing Encapsulation
48	MHRP	Mobile Host Routing Protocol
49	BNA	BNA
50	ESP	Encap Security Payload for IPv6
51	AH	Authentication Header for IPv6
52	I-NLSP	Integrated Net Layer Security TUBA
53	SWIPE	IP with Encryption
54	NARP	NBMA Address Resolution Protocol
55	MOBILE	IP Mobility
56	TLSP	Transport Layer Security Protocol using Kryptonet key management
57	SKIP	SKIP
58	IPv6-ICMP	ICMP for IPv6
59	IPv6-NoNxt	No Next Header for IPv6
60	IPv6-Opts	Destination Options for IPv6
61		any host internal protocol
62	CFTP	CFTP
63		any local network
64	SAT-EXPAK	SATNET and Backroom EXPAK
65	KRYPTOLAN	Kryptolan
66	RVD	MIT Remote Virtual Disk Protocol
67	IPPC	Internet Pluribus Packet Core
68		any distributed file system
69	SAT-MON	SATNET Monitoring
70	VISA	VISA Protocol
71	IPCV	Internet Packet Core Utility
72	CPNX	Computer Protocol Network Executive
73	CPHB	Computer Protocol Heart Beat
74	WSN	Wang Span Network
75	PVP	Packet Video Protocol
76	BR-SAT-MON	Backroom SATNET Monitoring
77	SUN-ND	SUN ND PROTOCOL-Temporary
78	WB-MON	WIDEBAND Monitoring
79	WB-EXPAK	WIDEBAND EXPAK
80	ISO-IP	ISO Internet Protocol
81	VMTP	VMTP
82	SECURE-VMTP	SECURE-VMTP
83	VINES	VINES
84	TTP	TTP
85	NSFNET-IGP	NSFNET-IGP
86	DGP	Dissimilar Gateway Protocol
87	TCF	TCF
88	EIGRP	EIGRP
89	OSPFIGP	OSPFIGP
90	Sprite-RPC	Sprite RPC Protocol
91	LARP	Locus Address Resolution Protocol
92	MTP	Multicast Transport Protocol

Decimal	Keyword	Protocol
93	AX.25	AX.25 Frames
94	IPIP	IP-within-IP Encapsulation Protocol
95	MICP	Mobile Internetworking Control Pro.
96	SCC-SP	Semaphore Communications Sec. Pro.
97	ETHERIP	Ethernet-within-IP Encapsulation
98	ENCAP	Encapsulation Header
99		any private encryption scheme
100	GMTP	GMTP
101	IFMP	Ipsilon Flow Management Protocol
102	PNNI	PNNI over IP
103	PIM	Protocol Independent Multicast
104	ARIS	ARIS
105	SCPS	SCPS
106	QNX	QNX
107	A/N	Active Networks
108	IPComp	IP Payload Compression Protocol
109	SNP	Sitara Networks Protocol
110	Compaq-Peer	Compaq Peer Protocol
111	IPX-in-IP	IPX in IP
112	VRRP	Virtual Router Redundancy Protocol
113	PGM	PGM Reliable Transport Protocol
114		any 0-hop protocol
115	L2TP	Layer Two Tunneling Protocol
116	DDX	D-II Data Exchange (DDX)
117	IATP	Interactive Agent Transfer Protocol
118	STP	Schedule Transfer Protocol
119	SRP	SpectraLink Radio Protocol
120	UTI	UTI
121	SMP	Simple Message Protocol
122	SM	SM
123	PTP	Performance Transparency Protocol
124	ISIS over IPv4	
125	FIRE	
126	CRTP	Combat Radio Transport Protocol
127	CRUDP	Combat Radio User Datagram
128	SSCOPMCE	
129	IPLT	
130	SPS	Secure Packet Shield
131	PIPE	Private IP Encapsulation within IP
132	SCTP	Stream Control Transmission Protocol
133	FC	Fibre Channel
134-254		Unassigned
255		Reserved

Appendix B: Mapping Operating Systems for answering/discarding ICMP query message types

Operating System	Info. Request	Time Stamp Request	Address Mask Request	Address Mask Request Frag.	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Linux Kernel 2.4.x	-	+	-	-	255	64
Linux Kernel 2.2.x	-	+	-	-	255	64
Linux Kernel 2.0.x					64	64
FreeBSD 4.0	-	+	-	-	255	255
FreeBSD 3.4	-	+	-	-	255	255
OpenBSD 2.7	-	+	-	-	255	255
OpenBSD 2.6	-	+	-	-	255	255
NetBSD	-	+	-	-	255	
BSDI BSD/OS 4.0	-	+	-	-	255	
BSDI BSD/OS 3.1	-	+	-	-	255	
Solaris 2.5.1	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.6	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.7	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.8	-	+	+	+ (0.0.0.0)	255	255
HP-UX v10.20	+	+	-	-	255	255
HP-UX v11.0	-	-	+	+ (0.0.0.0)	255	
Compaq Tru64 v5.0	+	+	-	-	64	
Irix 6.5.3	-	+	-	-	255	
Irix 6.5.8	-	+	-	-	255	
AIX 4.1	+	+	-	-	255	
AIX 3.2	+	+	-	-	255	
ULTRIX 4.2 – 4.5	+	+	+	+	255	
OpenVMS v7.1-2	+	+	+	+	255	
Novell Netware 5.1 SP1	-	-	-	-	128	
Novell Netware 5.0	-	-	-	-	128	
Novell Netware 3.12	-	-	-	-	128	
Windows 95	-	-	+	+	32	32
Windows 98	-	+	+	+	128	32
Windows 98 SE	-	+	+	+	128	32
Windows ME	-	+	-	-	128	32
Windows NT 4 WRKS SP 3	-	-	+	+	128	32
Windows NT 4 WRKS SP 6a	-	-	-	-	128	32
Windows NT 4 Server SP4	-	-	-	-	128	32
Windows 2000 Professional	-	+	-	-	128	128
Windows 2000 Server	-	+	-	-	128	128

Networking Devices	Info. Request	Request	Address Mask Request	Address Mask Request Frag.	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Cisco Catalyst 5505 with OSS v4.5	+	+	+	-	60	60
Cisco Catalyst 2900XL with IOS 11.2	+	+	-	-	255	
Cisco 3600 with IOS 11.2	+	+	-	-	255	
Cisco 7200 with IOS 11.3	+	+	-	-	255	255
Intel Express 8100 ISDN Router (*)	-	-	+	+	64	

Appendix C: ICMP Query Message Types with Code field !=0

Operating System	Info. Request	Time Stamp Request	Address Mask Request	ECHO Request
Linux Kernel 2.4.x	-	+ (0)	-	+ (!=0)
Linux Kernel 2.2.x	-	+ (0)	-	+ (!=0)
FreeBSD 4.0	-	+ (!=0)	-	+ (!=0)
FreeBSD 3.4	-	+ (!=0)	-	
OpenBSD 2.7	-	+ (!=0)	-	+ (!=0)
OpenBSD 2.6	-	+ (!=0)	-	+ (!=0)
NetBSD	-	+ (!=0)	-	+ (!=0)
BSDI BSD/OS 4.0	-	+ (!=0)	-	+ (!=0)
BSDI BSD/OS 3.1	-	+ (!=0)	-	+ (!=0)
Solaris 2.5.1	*	+ (!=0)	+ (!=0)	+ (!=0)
Solaris 2.6	*	+ (!=0)	+ (!=0)	+ (!=0)
Solaris 2.7	*	+ (!=0)	+ (!=0)	+ (!=0)
Solaris 2.8	*	+ (!=0)	+ (!=0)	+ (!=0)
HP-UX v10.20	+ (!=0)	+ (!=0)	-	
HP-UX v11.0	-	-	+ (!=0)	+ (!=0)
Compaq Tru64 v5.0	+ (!=0)	+ (!=0)	-	+ (!=0)
Irix 6.5.3	-	+ (!=0)	-	+ (!=0)
Irix 6.5.8	-	+ (!=0)	-	+ (!=0)
AIX 4.1	+ (!=0)	+ (!=0)	-	+ (!=0)
Aix 3.2	+ (!=0)	+ (!=0)	-	
ULTRIX 4.2 - 4.5	+ (!=0)	+ (!=0)	+ (!=0)	+ (!=0)
OpenVMS v7.1-2	+ (!=0)	+ (!=0)	+ (!=0)	+ (!=0)
Novell Netware 5.1 SP1	-	-	-	+ (!=0)
Novell Netware 5.0	-	-	-	+ (!=0)
Novell Netware 3.12	-	-	-	+ (!=0)
Windows 95	-	-	+	+ (0)
Windows 98	-	- (CHANGE)	+	+ (0)
Windows 98 SE	-	- (CHANGE)	+	+ (0)
Windows ME	-	- (CHANGE)	-	+ (0)
Windows NT 4 WRKS SP 3	-	-	+	+ (0)
Windows NT 4 WRKS SP 6a	-	-	-	+ (0)
Windows NT 4 Server SP4	-	-	-	+ (0)
Windows 2000 Professional	-	- (CHANGE)	-	+ (0)
Windows 2000 Server	-	- (CHANGE)	-	+ (0)

Networking Devices	Info. Request	Time Stamp Request	Address Mask Request	ECHO Request
Cisco Catalyst 5505 with OSS v4.5	+	+	+	+ (!0)
Cisco Catalyst 2900XL with IOS 11.2	+	+	-	+ (!0)
Cisco 3600 with IOS 11.2				+ (!0)
Cisco 7200 with IOS 11.3	+	+	-	+ (!0)
Intel Express 8100 ISDN Router (*)				

Appendix D: ICMP Query Message Types aimed at a Broadcast Address

Operating System	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast	Echo Request Broadcast
Linux Kernel 2.4.x				+
Linux Kernel 2.2.x	-	+	-	+
FreeBSD 4.0	-	-	-	-
FreeBSD 3.4	-	-	-	-
OpenBSD 2.7	-	-	-	-
OpenBSD 2.6	-	-	-	-
NetBSD				
BSDI BSD/OS 4.0				
BSDI BSD/OS 3.1				
Solaris 2.5.1	*	+	-	+
Solaris 2.6	*	+	-	+
Solaris 2.7	*	+	-	+
Solaris 2.8	*	+	-	+
HP-UX v10.20	+	+	-	+
Compaq Tru64 v5.0				
Irix 6.5.3				
Irix 6.5.8				
AIX 4.1				
AIX 3.2				
ULTRIX 4.2 – 4.5				
OpenVMS v7.1-2				
Novell Netware 5.1 SP1				
Novell Netware 5.0				
Novell Netware 3.12				
Windows 95				
Windows 98	-	-	-	-
Windows 98 SE	-	-	-	-
Windows ME	-	-	-	-
Windows NT 4 WRKS SP 3	-	-	-	-
Windows NT 4 WRKS SP 6a	-	-	-	-
Windows NT 4 Server SP4	-	-	-	-
Windows 2000 Professional	-	-	-	-
Windows 2000 Server	-	-	-	-

Networking Devices	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast	Echo Broadcast
Cisco Catalyst 5505 with OSS v4.5	+	+	+	+
Cisco Catalyst 2900XL with IOS 11.2	+	-	-	+
Cisco 3600 with IOS 11.2	+	-	-	
Cisco 7200 with IOS 11.3	+	-	-	+
Intel Express 8100 ISDN Router (*)	-	-	-	-

Appendix E: Precedence Bits Echoing with ICMP Query Request & Reply

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
Linux Kernel 2.4.x	Not Answering	!=0x00	Not Answering	!=0x00
Linux Kernel 2.2.x	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.1.1	Not Answering	!=0x00	Not Answering	
OpenBSD 2.7	Not Answering		Not Answering	!=0x00
OpenBSD 2.6	Not Answering		Not Answering	!=0x00
NetBSD	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering		Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00 -> 0x00	!=0x00 -> 0x00
Compaq Tru64 v5.0	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.3	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.2.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 3.2	!=0x00	!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5	0x00	0x00	0x00	0x00
OpenVMS v7.1-2	0x00	0x00	0x00	!=0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering			!=0x00
Windows 98 SE	Not Answering	0x00	0x00	!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Appendix F: ICMP Query Message Types with TOS! = 0

Operating System	Information Request With TOS!=0x00	Time Stamp Request With TOS!=0x00	Address Mask Request With TOS!=0x00	Echo Request With TOS!=0x00
Linux Kernel 2.4.x	Not Answering	!=0x00	Not Answering	!=0x00
Linux Kernel 2.2.x	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 3.4	Not Answering		Not Answering	
OpenBSD 2.7	Not Answering	!=0x00	Not Answering	!=0x00
OpenBSD 2.6	Not Answering	!=0x00	Not Answering	!=0x00
NetBSD	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering	!=0x00	Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented			
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00	!=0x00
Compaq Tru64 v5.0		!=0x00	Not Answering	!=0x00
Irix 6.5.3	Not Answering	!=0x00	Not Answering	!=0x00
Irix 6.5.8	Not Answering	!=0x00	Not Answering	!=0x00
AIX 4.1		!=0x00	Not Answering	!=0x00
AIX 3.2		!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5		0x00	0x00	0x00
OpenVMS v7.1-2		!=0x00	!=0x00	!=0x00
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x00	0x00	!=0x00
Windows 98 SE	Not Answering	0x00		!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Appendix G: Echoing the TOS Byte Unused bit

Operating System	Information Request	Time Stamp Request With Unused=1	Request With Unused=1	Echo Request With Unused=1
Linux Kernel 2.4.x	Not Answering	0x1	Not Answering	0x1
Linux Kernel 2.2.x	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.0	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.1.1	Not Answering	0x1	Not Answering	0x1
OpenBSD 2.7	Not Answering		Not Answering	
OpenBSD 2.6	Not Answering		Not Answering	
NetBSD	Not Answering		Not Answering	
BSDI BSD/OS 4.0	Not Answering		Not Answering	
BSDI BSD/OS 3.1	Not Answering		Not Answering	
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	0x1	0x1	0x1
Solaris 2.7	Not Implemented	0x1	0x1	0x1
Solaris 2.8	Not Implemented	0x1	0x1	0x1
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	0x1	0x1
Compaq Tru64 v5.0	0x1	0x1	Not Answering	0x1
AIX 4.3	0x1	0x1	Not Answering	0x1
AIX 4.2.1	0x1	0x1	Not Answering	0x1
AIX 4.1	0x1	0x1	Not Answering	0x1
AIX 3.2	0x1	0x1	Not Answering	0x1
ULTRIX 4.2 – 4.5	0x0	0x0	0x0	0x0
OpenVMS v7.1-2	0x1	0x1	0x1	0x1
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x0	0x0	0x1
Windows 98 SE	Not Answering	0x0	0x0	0x1
Windows ME	Not Answering	0x0	Not Answering	0x1
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	0x1
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	
Windows 2000 Professional	Not Answering	0x0	Not Answering	0x0
Windows 2000 Server	Not Answering	0x0	Not Answering	0x0

Appendix H: Using the Unused Bit

Operating System	Info. Request	Time Stamp Request	Address Mask Request	Echo Request
Linux Kernel 2.4.x	Not Answering	-	Not Answering	-
Linux Kernel 2.2.x	Not Answering	-	Not Answering	-
FreeBSD 4.0	Not Answering	-	Not Answering	-
FreeBSD 3.4	Not Answering	-	Not Answering	-
OpenBSD 2.7	Not Answering	-	Not Answering	-
OpenBSD 2.6	Not Answering	-	Not answering	-
NetBSD	Not Answering	-	Not Answering	-
BSDI BSD/OS 4.0	Not Answering	-	Not Answering	-
BSDI BSD/OS 3.1	Not Answering	-	Not Answering	-
Solaris 2.5.1	Not Answering	+	+	+
Solaris 2.6	Not Answering	+	+	+
Solaris 2.7	Not Answering	+	+	+
Solaris 2.8	Not Answering	+	+	+
HP-UX v10.20	-	-	Not Answering	-
HP-UX v11.0	Not Answering	Not Answering	+	+
Compaq Tru64 v5.0	-	-	Not Answering	-
Irix 6.5.3	Not Answering	-	Not Answering	-
Irix 6.5.8	Not Answering	-	Not Answering	-
AIX 4.1	-	-	Not Answering	-
AIX 3.2	-	-	Not Answering	-
ULTRIX 4.2 – 4.5	-	-	-	-
OpenVMS v7.1-2	-	-	-	-
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	-
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	-
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	-
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	-	-	-
Windows 98 SE	Not Answering	-	-	-
Windows ME	Not Answering		Not Answering	
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	-
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	-
Windows 2000 Professional	Not Answering	-	Not Answering	-
Windows 2000 Server	Not Answering	-	Not Answering	-

Appendix I: DF Bit Echoing

Operating System		Request	Address Mask Request	
Linux Kernel 2.4.x	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Linux Kernel 2.2.x	Not Answering	+ (- DF)	Not Answering	+ (- DF)
FreeBSD 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
FreeBSD 3.4	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.7	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.6	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
NetBSD	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 3.1	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Solaris 2.5.1	Not Answering			
Solaris 2.6	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.7	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.8	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	+ (+ DF)	+ (+ DF)
Compaq Tru64 v5.0		+ (+ DF)	Not Answering -	+ (+ DF)
Irix 6.5.3	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Irix 6.5.8	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
AIX 4.1		+ (+ DF)	Not Answering	+ (+ DF)
AIX 3.2		+ (+ DF)	Not Answering	+ (+ DF)
ULTRIX 4.2 – 4.5		+ (- DF)	+ (- DF)	+ (- DF)
OpenVMS v7.1-2		+ (+ DF)	+ (+ DF)	+ (+ DF)
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	+ (- DF)
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows 98 SE	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows ME	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows 2000 Professional	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows 2000 Server	Not Answering	+ (- DF)	Not Answering	+ (+ DF)

Appendix J: ICMP Error Message Echoing Integrity with ICMP Port Unreachable Error Message

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
Linux Kernel 2.4.x	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Linux Kernel 2.2.x	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
FreeBSD 4.0	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
FreeBSD 4.11	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
BSDI 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed. Now equals to ZERO!	Same
Sun Solaris 2.6	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.7	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.8[76]	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
HPUX 11.0	No -> Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Compaq Tru64	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
DG-UX 5.6	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!

[76] The DF Bit is set.

AIX 4.3 fp2, 4.3, 4.2.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Same
ULTRIX	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
OpenVMS	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
Microsoft windows 98 Mirosoft Windows 98SE	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows ME	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows NT 4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows 2000 Family	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same

Appendix K: Passive Fingerprinting Using ICMP Echo Requests with the '`ping`' Utility

Operating System	DF Bit Set?	IP ID Gap	IP Time-To-Live with request starting value	ICMP ID Field Value Starts with HEX / Decimal	Value	ICMP Sequence Number Initial Value	ICMP Sequence Number Gap	Payload - Content offset from the ICMP Header (bytes)	Payload - Content	Payload – Size (bytes)
Linux Kernel 2.2.x	No	1	64	According to other processes in the System	According to other processes in the System	0	100 / 256	8	Symbols & Signs	56
Linux Kernel 2.4.x	No	1	64			0	100 / 256	8		56
FreeBSD 4.1	No	1	255			0		8		56
FreeBSD 3.4	No	1	255	According to other processes in the System	According to other processes in the System	0		8	Symbols & Signs	56
OpenBSD 2.7	No		255					8		56
OpenBSD 2.6	No		255					8		56
NetBSD	No	1	255			0		8		56
BSDI BSD/OS 4.0	No		255					8		56
BSDI BSD/OS 3.1	No		255					8		56
Aix 4.1		1	255			0	1 / 1	8	Symbols & Signs	56
Solaris 2.5.1	Yes	1	255	According to other processes in the System	According to other processes in the System	0	1 / 1	8	Symbols & Signs	56
Solaris 2.6	Yes	1	255			0	1 / 1	8		56
Solaris 2.7	Yes	1	255			0	1 / 1	8		56
Solaris 2.8	Yes	1	255			0	1 / 1	8		56
Windows 95	No		32							
Windows 98	No	256	32			256	100 / 256	0	Alphabet	32
Windows 98 SE	No	256	32	200 / 512	Value Always = 512 Equals the number first assigned	256	100 / 256	0	Alphabet	32
Windows ME	No	1	32	300 / 768	Value Always = 768 Equals the number first assigned	256	100 / 256	0	Alphabet	32

Operating System	DF Bit Set?	IP ID Gap	IP Time-To-Live with request starting value	ICMP ID Field Value Starts with HEX / Decimal	ICMP ID Value	ICMP Sequence Number Initial Value	ICMP Sequence Number Gap	Payload - Content offset from the ICMP Header (bytes)	Payload - Content	Payload – Size (bytes)
Windows NT 4 Workstation SP3	No	256	32	100 / 256	Value Always = 256 Equals the number first assigned	256	100 / 256	0	Alphabet	32
Windows NT 4 Workstation SP6a	No	256	32	100 / 256	Value Always = 256 Equals the number first assigned	256	100 / 256	0	Alphabet	32
Windows 2000 Family	No	1	128	200 / 512	Value Always = 512 Equals the number first assigned	256	100 / 256	0	Alphabet	32
Windows 2000 Family with SP1	No	1	128	300 / 768	Value Always = 768 Equals the number first assigned	256	100 / 256	0	Alphabet	32

Appendix L: Host Based Security Prevention Methods

We can try to prevent some of the techniques demonstrated with this research using some Host based security configuration options.

K.1 Linux Kernel 2.4.x

The relevant configuration options available with Linux 2.4.x are:

- **`icmp_echo_ignore_all`**
  Controls if the Kernel answer for ICMP Echo requests.
  Default: Disabled (0)

- **`icmp_echo_ignore_broadcast`**
  "If either is set to true, then the kernel will ignore either all ICMP Echo requests sent to it or just those to broadcast/multicast addresses, respectively".
  Default: Disabled (0)

- **`icmp_ignore_bogus_error_responses`**
  "Some routers violate RFC 1122 by sending bogus responses to broadcast frames. Such violations are normally logged via a kernel warning. If this is set to TRUE, the kernel will not give such warnings, which will avoid log file clutter".
  Default: Disabled (0).

You can configure these configuration options using commands similar to the following (which will prevent the Linux based machine from answering an ICMP Echo requests):

RedHat 6.1, as root:
"*echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all*"
or put this line in the "`/etc/rc.d/rc.local`" file.

RedHat 6.2, as root:
Edit "`/etc/sysctl.conf`" and add the line: "`net.ipv4.icmp_echo_ignore_all=1`"

To see all parameters, as root:
```
# /sbin/sysctl -a
```

The problem with Linux Kernel 2.2.x / 2.4.x is that we cannot block ICMP Timestamp requests, or block ICMP Timestamp requests aimed at the broadcast address of the network the Linux based machine resides on.

K.2 Sun Solaris 8[77]

With Sun Solaris we have a lot more configuration options relevant to our research:

- **`ip_respond_to_echo_broadcast`**
  Control whether IPv4 responds to broadcast ICMPv4 echo request.
  Default: 1 (Enabled)

[77] Taken from "Solaris Tunable Parameters Reference Manual" for Sun Solaris 8. Available from http://docs.sun.com.

As root:
```
# ndd -set /dev/ip ip_respond_to_echo_broadcast 0
```

- **`ip_respond_to_timestamp_broadcast`**
  Control whether IPv4 responds to broadcast ICMPv4 timestamp request.
  Default: 1 (Enabled)

  As root:
  ```
  # ndd -set /dev/ip ip_respond_to_timestamp_broadcast 0
  ```

- **`ip_respond_to_address_mask_broadcast`**
  Control whether IPv4 responds to broadcast ICMPv4 address request.
  Default: 1 (Enabled)

  As root:
  ```
  # ndd /dev/ip ip_respond_to_address_mask_broadcast 0
  ```

- **`ip_respond_to_timestamp`**
  Controls if the Kernel answer for ICMP Timestamp request.
  Default: 1 (Enabled)

  As root:
  ```
  # ndd -set /dev/ip ip_respond_to_timestamp 0
  ```

The problem with Sun Solaris is that we can still send ICMP Address Mask requests and ICMP Echo requests destined the Sun Solaris based machine, and they will be answered.

More Advanced Configuration Options:

- **`ip_icmp_err_interval` and `ip_icmp_err_burst`**
  Control the rate of IP in generating IPv4 or IPv6 ICMP error messages. IP generates only up to `ip_icmp_err_burst` IPv4 or IPv6 ICMP error messages in any `ip_icmp_err_interval`. This parameter protects IP from denial of service attacks.

  Set `ip_icmp_err_interval` to 0 to disable IP to generate IPv4 or IPv6 ICMP error messages.

  Default:
  - 100 milliseconds for `ip_icmp_err_interval`
  - 10 for `ip_icmp_err_burst`

  Range:
  - 0 - 99,999 milliseconds for `ip_icmp_err_interval`
  - 1 - 99,999 for `ip_icmp_err_burst`

- **`ip_send_redirects`**
  Controls wether IPv4 sends out ICMPv4 redirect messages.
  Default: Enabled (1)

- **`ip_ignore_redirects`**
  Default: 0 (not ignore)

- **`ip_icmp_return_data_bytes`**
  When IPv4 or IPv6 sends an ICMPv4 or ICMPv6 error message, it includes the IP header of the packet that causes the error message. This parameter controls how many extra bytes of the packet beyond the IPv4 or IPv6 header to be included in the ICMPv4 or ICMPv6 error.
  Deafult: 64
  Range: 8 to 65.536

- **`ip_ire_pathmtu_interval`**
  The interval in milliseconds when IP flushes the path maximum transfer unit (PMTU) discovery information, and tries to rediscover PMTU.
  Default: 10 Minutes
  Range: 5 seconds to 277 hours

Other interesting parameters without any description in the Sun manuals are:

- `icmp_accept_clear_messages`
- `ip_def_ttl`
- `ip_broadcast_ttl`

K.2.1 How to set a TCP/IP parameter across reboots?

You should include the appropriate `ndd` command in a system startup script. You can use the following guidelines to create a system startup script to include `ndd` commands as outlined in the "Solaris Tunable Parameters Reference Manual":

- Create a script in the `/etc/init.d` directory and create links to it in the `/etc/rc2.d`, `/etc/rc1.d`, and `/etc/rcS.d` directories.
- The script should run between the existing `S69inet` and `S72inetsvc` scripts.
- Name the script with the `S70` or `S71` prefix. Scripts with the same prefix are run in some sequential way so it doesn't matter if there is more than one script with the same prefix.
- See the README file in the `/etc/init.d` directory for more information on naming run control scripts.

Appendix M: A Snort Rule Base for (more Advanced) Basic ICMP Traffic

The following generic ICMP basic Snort rule base is also available for download from: http://www.sys-security.com/archive/snort/icmp_rules/ICMP_basic_plus.

```
alert icmp any any -> any any (msg:"ICMP Echo Reply"; itype: 0; icode:
0;)
alert icmp any any -> any any (msg:"ICMP Echo Reply (Undefined Code!)";
itype: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 1)"; itype:
1; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Tupe 1)
(Undefined Code)"; itype: 1;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 2)"; itype:
2; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 2)"
(Undefined Code); itype: 2;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Network Unreachable)"; itype: 3; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Unreachable)"; itype: 3; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Protocol Unreachable)"; itype: 3; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Port
Unreachable)"; itype: 3; icode: 3;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Fragmentation Needed and DF bit was set)"; itype: 3; icode: 4;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Source Route Failed)"; itype: 3; icode: 5;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Destination Network Unknown)"; itype: 3; icode: 6;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Destination Host Unknown)"; itype: 3; icode: 7;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Source Host Isolated)"; itype: 3; icode: 8;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication with Destination Network is Administratively
Prohibited)"; itype: 3; icode: 9;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication with Destination Host is Administratively Prohibited)";
itype: 3; icode: 10;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Network Unreachable for Type of Service)"; itype: 3; icode: 11;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Unreachable for Type of Service)"; itype: 3; icode: 12;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication Administratively Prohibited)"; itype: 3; icode: 13;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Precedence Violation)"; itype: 3; icode: 14;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Precedence Cutoff in effect)"; itype: 3; icode: 15;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Undefined Code!)"; itype: 3;)
```

```
alert icmp any any -> any any (msg:"ICMP Source Quench"; itype: 4;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Source Quench (Undefined
Code!)"; itype: 4;)
alert icmp any any -> any any (msg:"ICMP Redirect (for Network or
Subnet)"; itype: 5; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Redirect (for Host)"; itype:
5; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Redirect (for TOS and
Network)"; itype: 5; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Redirect (for TOS and Host)";
itype: 5; icode: 3;)
alert icmp any any -> any any (msg:"ICMP Redirect (Undefined Code!)";
itype: 5;)
alert icmp any any -> any any (msg:"ICMP Alternate Host Address";
itype: 6; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Alternate Host Address
(Undefined Code!)"; itype: 6;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 7)"; itype:
7; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 7)
(Undefined Code!)"; itype: 7;)
alert icmp any any -> any any (msg:"ICMP Echo Request"; itype: 8;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Echo Request (Undefined
Code!)"; itype: 8;)
alert icmp any any -> any any (msg:"ICMP Router Advertisment"; itype:
9; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Router Advertisment (Undefined
Code!)"; itype:9 ;)
alert icmp any any -> any any (msg:"ICMP Router Selection"; itype: 10;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Router Selection (Undefined
Code!)"; itype: 10;)
alert icmp any any -> any any (msg:"ICMP Time-To-Live Exceeded in
Transit"; itype: 11; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Fragment Reassembly Time
Exceeded"; itype: 11; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Time Exceeded (Undefined
Code!)"; itype: 11;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 0
(unspecified Error)"; itype: 12; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 1
(Missing a Requiered Option)"; itype: 12; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 2 (Bad
Length)"; itype: 12; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem (Undefined
Code!)"; itype: 12;)
alert icmp any any -> any any (msg:"ICMP Timestamp Request"; itype: 13;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Timestamp Request (Undefined
Code!)"; itype: 13;)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply"; itype: 14;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply (Undefined
Code!)"; itype: 14;)
```

```
alert icmp any any -> any any (msg:"ICMP Information Request"; itype:
15; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Information Request (Undefined
Code!)"; itype: 15;)
alert icmp any any -> any any (msg:"ICMP Information Reply"; itype: 16;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Information Reply (Undefined
Code!)"; itype: 16;)
alert icmp any any -> any any (msg:"ICMP Address Mask Request"; itype:
17; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Address Mask Request
(Undefined Code!)"; itype: 17;)
alert icmp any any -> any any (msg:"ICMP Address Mask Reply"; itype:
18; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Address Mask Reply (Undefined
Code!)"; itype: 18;)
alert icmp any any -> any any (msg:"ICMP Reserved for Security (Type
19)"; itype: 19; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Reserved for Security (Type
19) (Undefined Code!)"; itype: 19;)
alert icmp any any -> any any (msg:"ICMP Traceroute"; itype: 30; icode:
0;)
alert icmp any any -> any any (msg:"ICMP Traceroute (Undefined Code!";
itype: 30;)
alert icmp any any -> any any (msg:"ICMP Datagram Conversion Error";
itype: 31; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Datagram Conversion Error
(Undefined Code!)"; itype: 31;)
alert icmp any any -> any any (msg:"ICMP Mobile Host Redirect"; itype:
32; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Host Redirect
(Undefined Code!)"; itype: 32;)
alert icmp any any -> any any (msg:"ICMP IPV6 Where-Are-You"; itype:
33; icode: 0;)
alert icmp any any -> any any (msg:"ICMP IPV6 Where-Are-You (Undefined
Code!)"; itype: 33;)
alert icmp any any -> any any (msg:"ICMP IPV6 I-Am-Here"; itype: 34;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP IPV6 I-Am-Here (Undefined
Code!"; itype: 34;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Request";
itype: 35; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Request
(Undefined Code!"; itype: 35;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Reply";
itype: 36; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Reply
(Undefined Code!)"; itype: 36;)
alert icmp any any -> any any (msg:"ICMP SKIP"; itype: 39; icode: 0;)
alert icmp any any -> any any (msg:"ICMP SKIP (Undefined Code!"; itype:
39;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 0 (Reserved)";
itype: 40; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 1 (Unknown
Security Parameters Index)"; itype: 40; icode: 1;)
```

```
alert icmp any any -> any any (msg:"ICMP Photuris Code 2 (Valid
Security Parameters, But Authentication Failed)"; itype: 40; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 3 (Valid
Security Parameters, But Decryption Failed)"; itype: 40; icode: 3;)
alert icmp any any -> any any (msg:"ICMP Photuris (Undefined Code!)";
itype: 40;)
alert icmp any any -> any any (msg:"ICMP Unknown Type";)
```

For corrections/additions/suggestions for this research paper please send email to ofir@sys-security.com.

Further Information and updates will be posted to http://www.sys-security.com.

Thank you for reading

Ofir Arkin

Founder

The Sys-Security Group

http://www.sys-security.com
ofir@sys-security.com