X remote ICMP based OS fingerprinting techniques

Ofir Arkin

ofir@sys-security.com

Founder

The Sys-Security Group

http://www.sys-security.com

Fyodor Yarochkin

fygrave@tigerteam.net

August 2001

Contents

Abstract

In this paper ICMP based remote OS fingerprinting techniques are discussed. A logic tree and a tool which deploys the tree logic are explained.

1 Introduction - What is X?

X is a logic which combines various remote active operating system fingerprinting methods using the ICMP protocol, which were discovered during the "ICMP Usage in Scanning" research project, into a simple, fast, efficient and a powerful way to detect an underlying operating system a targeted host is using.

Xprobe is a tool written and maintained by Fyodor Yarochkin (fygrave@tigerteam.net) and Ofir Arkin (ofir@sys-security.com) that automates X.

1.1 Why X?

X is a very accurate logic.

Xprobe is an alternative to some tools which are heavily dependent upon the usage of the TCP protocol for remote active operating system fingerprinting. This is especially true when trying to identify some Microsoft based operating systems, when TCP is the protocol being used with the fingerprinting process. Since the TCP implementation with Microsoft Windows 2000 and Microsoft Windows ME, and with Microsoft Windows NT 4 and Microsoft Windows 98/98SE are so close, usually when using the TCP protocol with a remote active operating systems fingerprinting process we are unable to differentiate between these Microsoft based operating system groups. And this is only an example.

As we will demonstrate the number of datagrams we need to send and receive in order to remotely fingerprint a targeted machine with X is small. Very small. In fact we can send one datagram and receive one reply and this will help us identify up to eight different operating systems (or groups of operating systems). The maximum datagrams the tool will send is four. This is the same number of replies we will need. This makes Xprobe very fast as well.

Xprobe probes can be very stealthy. Xprobe does not send any malformed datagrams to detect a remote OS type, unlike the common fingerprinting methods. Xprobe analyzes the remote OS *TCP/IP* stack responses for valid packets. Heaps of such packets appear in an average network on daily basis and very few IDS systems [1] are tuned to detect such traffic (and those which are, presumably are very badly configured).

Usually when people see the types of datagrams being used by Xprobe, they will think that what have happened was a simple Host Detection attempt, while in fact the replying machines were not only detected, but their underlying operating systems were revealed as well.

In the future Xprobe will be using actual application data with its probes. This will help in disguising the real intentions of the probes.

X might change the traditional intelligence gathering approach. With the traditional approach we need to detect the availability of a Host (using a Host Detection method), find a service it is running (using port scanning), and than identify the underlying operating system (with a remote active operating system fingerprinting tool). If the targeted machine is running a service that is known to be vulnerable it may allow a malicious computer attacker to execute a remote exploit in order to gain unauthorized access to the targeted machine.

With X we combine the host detection stage with the operating system detection stage. With maximum of four datagrams initiated from the malicious computer attacker's machine, we are able to determine if a certain machine is running an operating system where certain vulenrabilities might be presented (and attempted to be exploited). For example, a Microsoft Windows 2000 based operating system can be identified with four datagrams traversing over the network in total (two sent and two received).

Considering the amount of default installations of Microsoft Windows 2000 based systems in the network (with a vulnerable version of IIS 5.0 up and running) a malicious computer attacker might try to compromise the targeted machine with his third packet sent. This is especially true when our target is a web server (targeting www.mysite.com for example).

[1] Zero payload and a few other things, currently deployed in Xprobe can be used to tune your IDS to detect Xprobe probes currently. We will deal with these artifacts once the core system would be in more or less stable state, for the moment it is a low priority

2 Remote Active Operating System Fingerprinting Methods Used By X

X takes advantage of several remote active operating system fingerprinting methods discovered during the "ICMP Usage in Scanning"[1] research project.

2.1 ICMP Error Message Quoting Size

Each ICMP error message includes the 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[2].

Most of the operating systems will quote the offending packet's IP Header and the first 8 data bytes of the datagram that triggered the error. Several operating systems and networking devices will echo more than 8 data bytes.

Which operating systems will quote more?

Linux based on Kernel 2.0.x/2.2.x/2.4.x, Sun Solaris 2.x, HPUX 11.x, MacOS 7.x-9.x (10.x not checked), Nokia boxes, Foundry Switches (and other OSs and several Networking Devices) are good examples.

2.2 ICMP Error Message Echoing Integrity

Each ICMP error message includes the 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.

When sending back an ICMP error message, some stack implementations may alter the offending packet's IP header and the underlying protocol's data, which is echoed back with the ICMP error message.

If a malicious computer attacker examines the types of alternation*s* that have been made to the offending packet's IP header and the underlying protocol data, 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 decreased.

With X we will take advantage of ICMP Port Unreachable error messages triggered by UDP datagrams sent to close UDP ports. We will be examining several IP Header and UDP related field values of the offending packet being echoed with the ICMP Error message, for some types of alternation$s$.

IP Total Length Field - Some operating system IP stacks will add 20 bytes to the original IP total length field value of the offending packet, with the one echoed with the IP header of the offending packet in the ICMP Error message. Some other operating system IP stacks will decrease 20 bytes from the original IP total length field value of the offending packet, with the one echoed with the IP header of the offending packet in the ICMP Error message. And some other operating system IP stacks will echo correctly this field value.

IPID - Some operating system IP stacks will not echo the IPID field value correctly with their ICMP Error messages. They will change the bit order with the value echoed. Other operating system IP stacks will echo correctly this field value.

3Bits Flags and Offset Fields - Some operating system IP stacks will not echo the 3Bits Flags and Offset fields value correctly with their ICMP Error messages. They will change the bit order with these fields. Other operating system IP stacks will echo correctly this field value.

IP Header Checksum - Some operating system IP stacks will miscalculate the IP Header checksum of the offending packet echoed back with the ICMP error message. Some operating system IP stacks will zero out the IP Header checksum of the offending packet echoed back with the ICMP error message. Other operating system IP stacks will echo correctly this field value.

UDP Header Checksum - Some operating system IP stacks will miscalculate the UDP Header checksum of the offending packet echoed back with the ICMP error message. Some operating system IP stacks will zero out the UDP Header checksum of the offending packet echoed back with the ICMP error message. Other operating system IP stacks will echo correctly this field value.

Some operating system stacks will not echo correctly several field values with the same ICMP Error Message, and not just one. This will enable us to use multiple echoing integrity tests with just one ICMP Error messages sent by a targeted machine.

2.3 Precedence Bits Issues with ICMP Error Messages

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

The 'TOS Byte' consists of three fields.

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

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' (must be zero), is unused and must be zero. Routers and hosts ignore this last field. This field is 1 bit long. [2]

RFC 1812 Requirements for IP Version 4 Routers[4]:

"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".

Linux Kernel 2.0.x, 2.2.x, 2.4.x will act as routers and will set their Precedence bits field value to 0xc0 with ICMP error messages. Networking devices

[2]The TOS Bits and MBZ fields are being replaced by the DiffServ mechanism for QoS

that will act the same will be Cisco routers based on IOS 11.x-12.x and Foundry Networks switches.

2.4 DF Bit Echoing with ICMP Error Messages

What will happen if we will set the DF bit with an offending packet that will trigger an ICMP error message from a targeted machine? Will the DF Bit be set in the ICMP error message IP Header?

2.5 IP Identification Field Value with ICMP Query Messages with Linux Kernel 2.4.x based machines

Linux machines based on Kernel 2.4.0-2.4.4 will set the IP Identification field value with their ICMP query request and reply messages to a value of zero.

This was later fixed with Linux Kernels 2.4.5 and up.

2.6 The IP Time-To-Live Field Value with ICMP Messages

"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 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 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 datagram lifetime, so infinite loops of undelivered datagrams will not jam the Internet.

Having a bound to the datagram 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 criterion when we are querying other host(s) or listening to traffic (sniffing).

TTL-based fingeprinting requires a TTL distance to the done to be precalculated in advance (unless a fingerprinting of a local network based system is performed system).

The ICMP Error messages will use values used by ICMP query request messages.

2.7 Using Code Field Values Different Than Zero with ICMP Echo Requests

When an ICMP code field value different than zero (0) is sent with an ICMP Echo request message (type 8), operating systems that will answer our query with an ICMP Echo reply message that are based on one of the Microsoft based operating systems will send back an ICMP code field value of zero with their ICMP Echo Reply. Other operating systems (and networking devices) will echo back the ICMP code field value we were using with the ICMP Echo Request.

The Microsoft based operating systems acts in contrast to RFC 792[5] guidelines which instruct the answering operating systems to only change the ICMP type to Echo reply (type 0), recalculate the checksums and send the ICMP Echo reply away.

2.8 TOS Echoing

RFC 1349 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"[6].

  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.

Some operating systems will ignore RFC 1349 when sending ICMP echo reply messages, and will not send the same value in the TOS field as was used in the corresponding ICMP request message.

2.9 DF Bit Echoing With ICMP Query Messages

What will happen if we will set the DF bit with ICMP query request messages? Will the DF Bit be set with the ICMP query reply message?

2.10 The ?Who Answer What?? Approach

With this method we map which operating systems answer for which ICMP Query message request.

We use:

- ICMP Echo request
- ICMP Timestamp request
- ICMP Information request
- ICMP Address Mask request

3 How does X works?

The logic was first built as a static decision tree. It was intended to be a proof-of-concept logic using the various remote active operating system fingerprinting methods using the ICMP protocol which were discovered during the ICMP research project[3].

Understanding the Logic

The most important ability we have is to isolate certain operating systems and groups of operating systems according to a specific active operating system fingerprinting method.

We initiate the process with a UDP datagram sent to a definitely closed (3132 by default) UDP destination port.

Why are we using a UDP datagram?

We will take advantage of multiple operating system fingerprinting differences available with an ICMP Destination Unreachable Port Unreachable error messages. To trigger the ICMP Port Unreachable error message we use the UDP protocol.

Detecting the presence of a filtering device

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

[3]See the next chapter for future plans and enhancements

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. Thos, we will not receive any reply back.

Based on the fact that sending a UDP datagram to a closed UDP port should elicit an ICMP Port Unreachable error message, we will send one UDP datagram to a definitely closed UDP port we have chosen. Then:

- 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.

What is the definition of a "definitely closed"

We can try to identify a number of UDP ports that are not being used by any application. We can look at the IANA (Internet Assigned Numbers Authority) list for port numbers located at: `http://www.isi.edu/in-notes/iana/assignments/port-numbers` and choose some.

We can build a pool of numbers randomly picked at each time we use the logic.

We can also choose UDP ports that are likely to be closed, with services rarely used.

Combining other operating system fingerprinting tests with our UDP query

We can combine several active operating system fingerprinting methods into one query. We will send our UDP datagram with the DF Bit set. So the "DF Bit echoing with ICMP Error Messages" active operating system fingerprinting method will be used with the logic as well, not requiring additional ICMP or other protocol queries.

The UDP datagram query will be carrying 70 data bytes. It will allow us to test, if an ICMP Port Unreachable Error message is received, the amount of data being echoed by the targeted machine.

Assuming we did not received any reply back from a targeted IP address, we will consider it being protected by a filtering device.

After receiving an ICMP Port Unreachable error message from our targeted IP address - The game begins.

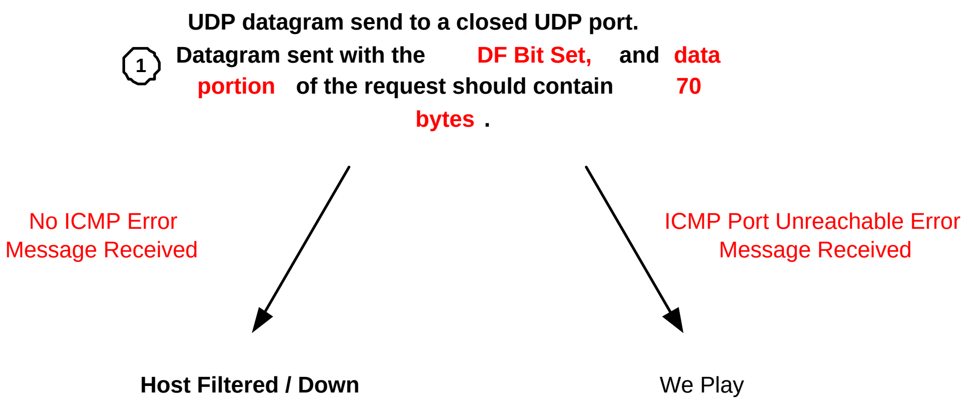
UDP datagram send to a closed UDP port.
1
Datagram sent with the DF Bit Set, and data portion of the request should contain 70 bytes .
No ICMP Error Message Received
ICMP Port Unreachable Error Message Received
Host Filtered / Down
We Play

Figure 1: In the Beginning

We will be examining the Precedence Bits field value received with the ICMP Port Unreachable Error message.

If this field value is equal to 0xc0 the questionable IP address is probably a Linux Kernel 2.0.x/ 2.2.x/2.4.x based machine, a Cisco based router running IOS version 11.x-12.x, or an Extreme Networks Switch.

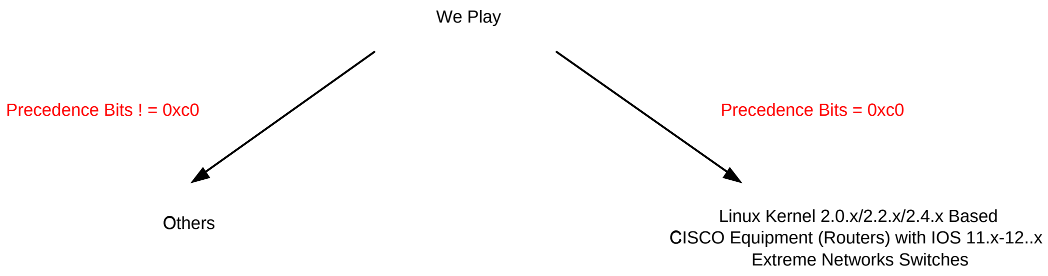
We Play
Precedence Bits ! = 0xc0
Precedence Bits = 0xc0
Others
Linux Kernel 2.0.x/2.2.x/2.4.x Based
CISCO Equipment (Routers) with IOS 11.x-12..x
Extreme Networks Switches

Figure 2: First Parameter to Look At

We need to use another check criterion to distinguish between the Linux Kernel 2.0.x/2.2.x/2.4.x based IP addresses to the IP addresses of the networking devices.

We will use the ICMP Error Quoting Size fingerprinting method.

Each ICMP error message includes the 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 packet's IP Header and the first 8 data bytes of the datagram that triggered the error. Several operating systems and networking devices will echo more than 8 data bytes.

One group of operating systems that will echo more than 8 data bytes of the offending packet's data will be Linux Kernel 2.0.x/2.2.x/2.4.x based machines.

The Networking devices that will set their precedence bits value to 0xc0 with their ICMP Port Unreachable error messages will echo only the first 8 data bytes from the offending packet's data.

This will enable us to divide between Linux Kernel 2.0.x/2.2.x/2.4.x based IP addresses to IP addresses assigned to Cisco Routers using IOS 11.x-12.x or Extreme Networks switches.

Using an Echoing Integrity problem with Extreme Networks switches (UDP Header being echoed will be zeroed) we will be dividing between the switches and the Cisco routers based on IOS 11.x-12.x

Back to the Linux branch

We are now interested in dividing the Linux Kernel 2.0.x/2.2.x/2.4.x based group. We will use the fact that Linux 2.0.x based machines will use 64 as their initial IP Time-to-Live field value with ICMP error messages, while Linux 2.2.x/2.4.x based machines will use 255 as their initial IP Time-to-Live field value with ICMP error messages.

How can we distinguish between Linux based machines running Kernel 2.2.x or Kernel 2.4.x?

With Linux Kernel 2.4.0-2.4.4 the IP ID field value with ICMP queries (and replies) will be always equal to zero. This was later fixed with Linux Kernel 2.4.5 and up, as seen with net/ipv4/ip_output.c:

```
- u16 id = 0;
+ u16 id;
```

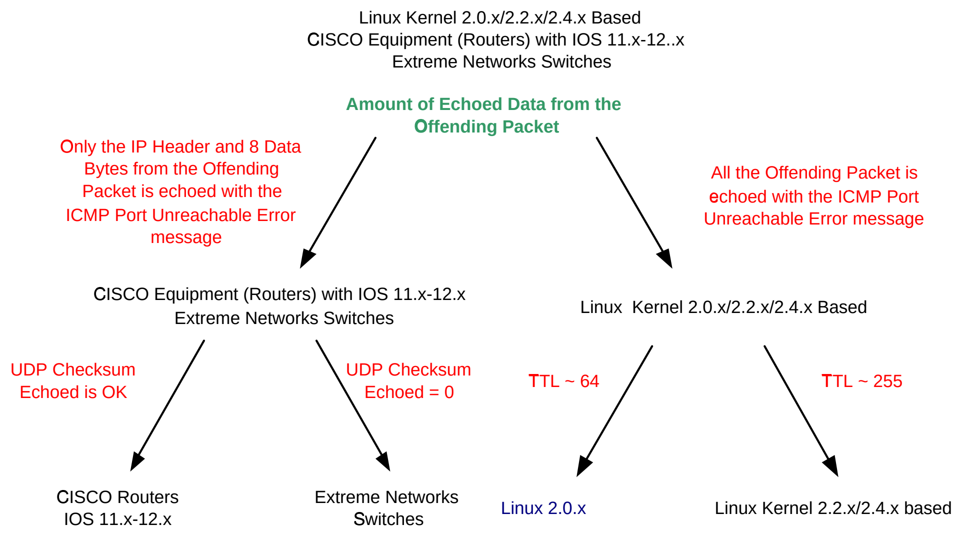
Linux Kernel 2.0.x/2.2.x/2.4.x Based
CISCO Equipment (Routers) with IOS 11.x-12..x
Extreme Networks Switches
Amount of Echoed Data from the Offending Packet
Only the IP Header and 8 Data Bytes from the Offending Packet is echoed with the ICMP Port Unreachable Error message
All the Offending Packet is echoed with the ICMP Port Unreachable Error message
CISCO Equipment (Routers) with IOS 11.x-12.x
Extreme Networks Switches
Linux Kernel 2.0.x/2.2.x/2.4.x Based
UDP Checksum Echoed is OK
UDP Checksum Echoed = 0
TTL ~ 64
TTL ~ 255
CISCO Routers IOS 11.x-12.x
Extreme Networks Switches
Linux 2.0.x
Linux Kernel 2.2.x/2.4.x based

Figure 3: Digging in the Precedence = 0xc0 branch

```
..
+ id = (sk ? sk->protinfo.af_inet.id++ : 0);
```

This will help us divide IP addresses which are based on Linux Kernel 2.2.x/2.4.5 (and up) from IP addresses that are based on Linux Kernel 2.4.0-2.4.4.

In order to use this method we will send ICMP Echo requests to the questionable IP addresses, waiting for an ICMP Echo reply to arrive.

The IP addresses producing an ICMP Echo reply with an IP ID field value of zero will be Linux Kernel 2.4.0-2.4.4 based, while the rest of the IP addresses will be Linux Kernel 2.2.x/2.4.5 based.

No reply is received for the ICMP Echo Request

We can than state that the questionable IP addresses are Linux Kernel 2.2.x or Linux Kernel 2.4.x based, but a filtering device prevented us from concluding.

The Other Side of the Moon

We will now focus on IP addresses which produce an ICMP error message with a precedence bits value of zero.

The second test we will put those IP addresses ICMP Error messages through

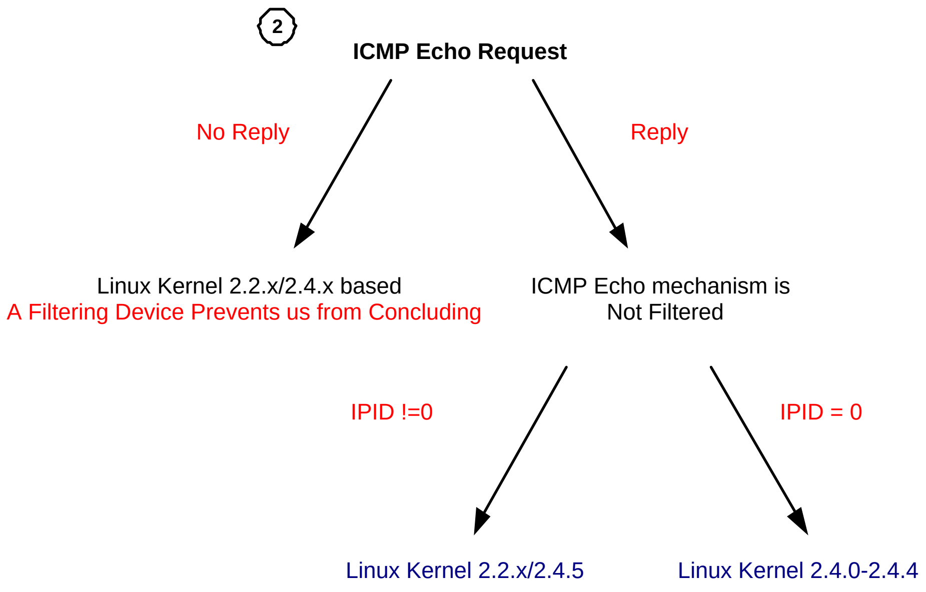
2
ICMP Echo Request
No Reply
Reply
Linux Kernel 2.2.x/2.4.x based
A Filtering Device Prevents us from Concluding
ICMP Echo mechanism is Not Filtered
IPID !=0
IPID = 0
Linux Kernel 2.2.x/2.4.5
Linux Kernel 2.4.0-2.4.4

Figure 4: Finding Linux Kernel 2.4.0-2.4.4

will be "How much echoed data of the offending packet is carried with the ICMP Port Unreachable Error Messages?"

We can divide the IP stacks into three groups:

- IP stacks that will echo 8 data bytes from the offending packet's data
- IP stacks that will echo 64 data bytes from the offending packet's data
- IP stacks that will echo more than 64 bytes from the offending packet's data

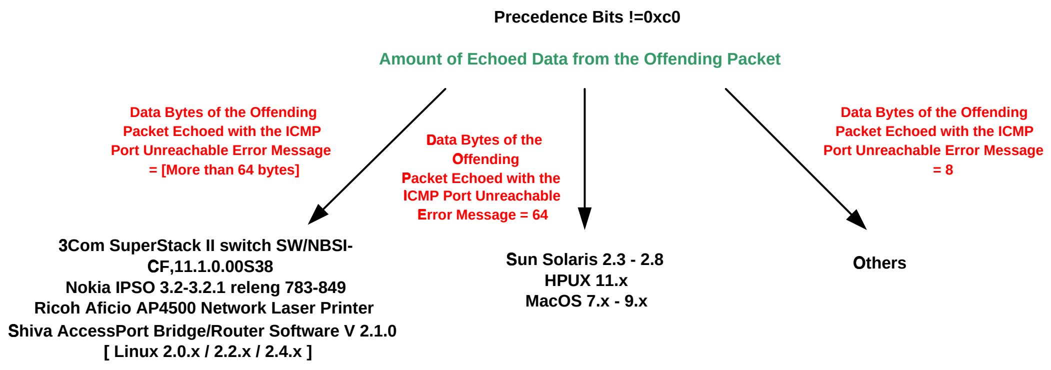
Precedence Bits !=0xc0
Amount of Echoed Data from the Offending Packet
Data Bytes of the Offending Packet Echoed with the ICMP Port Unreachable Error Message = [More than 64 bytes]
Data Bytes of the Offending Packet Echoed with the ICMP Port Unreachable Error Message = 64
Data Bytes of the Offending Packet Echoed with the ICMP Port Unreachable Error Message = 8
3Com SuperStack II switch SW/NBSI-CF,11.1.0.00S38
Nokia IPSO 3.2-3.2.1 releng 783-849
Ricoh Aficio AP4500 Network Laser Printer
Shiva AccessPort Bridge/Router Software V 2.1.0
[Linux 2.0.x / 2.2.x / 2.4.x]
Sun Solaris 2.3 - 2.8
HPUX 11.x
MacOS 7.x - 9.x
Others

Figure 5: The Other Side of the Moon

The 64 data bytes group

The operating systems that will echo, by default, 64 bytes from the offending packet's data portion are Sun Solaris 2.3-2.8, HPUX 11.x, and MacOS 7.x-9.x (10.x not checked).

We need another check criterion in order to differentiate between the operating systems belonging to this group.

We will query the questionable IP addresses with an ICMP timestamp request. Since Sun Solaris based machines will produce an ICMP Timestamp reply for this request, while the HPUX 11.x and MacOS 7.x-9.x will not, we will be able to divide this group.

If the ICMP Timestamp mechanism is blocked by a filtering device, all the questionable IP addresses will turn as HPUX 11.x/MacOS 7.x-9.x.

This is also true if we try to use host based security features that comes with Sun Solaris based machines. While we are able to configure a Sun Solaris machine to ignore ICMP timestamp requests, we are unable to configure it to ignore ICMP echo requests.

Since MacOS 7.x-9.x and HPUX 11.x IP stacks share similar behavior, we are unable to differentiate between them. [4].

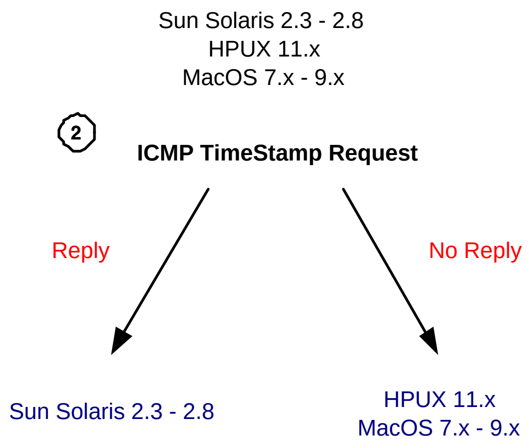

Figure 6: Differentiating Between Sun Solaris and HPUX 11.x/MacOS 7.x-9.x based machines

The More than 64 data bytes group

[4]If an HPUX 11.x based machine is configured to use the PMTU discovery process using ICMP Echo requests we might be able to differentiate between HPUX 11.x based machines to MacOS 7.x-9.x based machines

The operating systems and networking devices that will echo, by default, more than 64 bytes from the offending packet's data are 3Com SuperStack II switch SW/NBSI-CF 11.1.0.00S38, Nokia IPSO 3.2-3.2.1 releng 783-849, Ricoh Aficio AP4500 Network Laser Printer, and Shiva Access Port Bridge/Router Software V 2.1.0.

If, for some reason, the precedence bits field value will not be set to 0xc0 hex with Linux Kernel 2.0.x / 2.2.x / 2.4.x based machines, than these hosts will fall into this category as well.

The 8 data bytes group

Most of the IP stacks will echo only 8 bytes from the offending packet's data.

We will try to use, again, a fingerprinting test that will be conclusive with its results. One such test is an Echoing Integrity test examining the IP total length field value of the offending packet echoed with the ICMP Error message.

Here, again, we will have three main groups of IP stacks (and operating systems):

- The IP Total Length field value being echoed correctly
- The IP Total Length field value being echoed is 20 bytes less than the original value
- The IP Total Length field value being echoed is 20 bytes bigger than the original value

The IP Total Length field value is 20 bytes less than the original value

We will focus, first, on the group of operating systems that sets the value of the echoed IP total length field value to a value 20 bytes less than the original value. This group includes the following operating systems and networking devices: OpenBSD 2.6-2.9, NFR IDS appliance, Apollo Domain/OS SR10.4, Extreme Networks Switches, Network Systems router NS6614 (NSC 6600 series), and Cabletron Systems SSR 8000 System Software, Version 3.1.B.16.

We need to use another check criterion to divide this group.

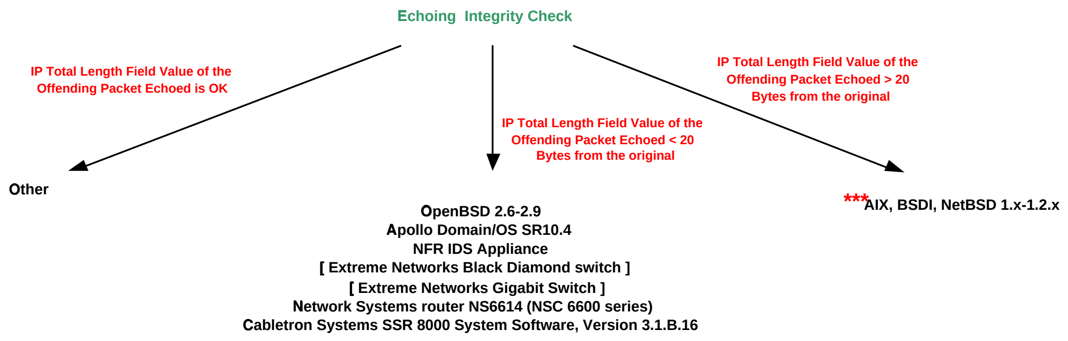
Echoing Integrity Check
IP Total Length Field Value of the Offending Packet Echoed is OK
IP Total Length Field Value of the Offending Packet Echoed < 20 Bytes from the original
IP Total Length Field Value of the Offending Packet Echoed > 20 Bytes from the original
Other
OpenBSD 2.6-2.9
Apollo Domain/OS SR10.4
NFR IDS Appliance
[Extreme Networks Black Diamond switch]
[Extreme Networks Gigabit Switch]
Network Systems router NS6614 (NSC 6600 series)
Cabletron Systems SSR 8000 System Software, Version 3.1.B.16
***AIX, BSDI, NetBSD 1.x-1.2.x

Figure 7: Echoing Integrity Problems: IP Total Length Field Value

We will use another Echoing Integrity problem. This time we will be examining the UDP checksum field value echoed with the ICMP error message.

With the group of operating systems and networking devices listed above, we will have three different values being echoed:

- The correct field value
- Zero
- ...And a faulty field value

Using this Echoing Integrity problem will allow us to isolate OpenBSD 2.6-2.9 based operating systems and Apollo domain/OS SR10.4 from the rest of the group according to the next diagram:

Since the Apollo Domain/OS echoes back a faulty IP Header checksum field value as well, we can isolate it, and most important - IP addresses using OpenBSD 2.6-2.9.

The IP Total Length field value is 20 bytes higher than the original value

With this group we will find operating systems such as: AIX 3.x, 4.x; BSDI 4.x, 3.x, 2.x; NetBSD 1.0, 1.1, 1.2; and MacOS X 1.0, 1.1, 1.2.

The way to divide this group is by looking at echoing integrity problems with the ICMP Error message these operating systems produced for our offending UDP datagram[5].

[5]Please note that the DF bit we refer to here is the one with the IP Header of the ICMP

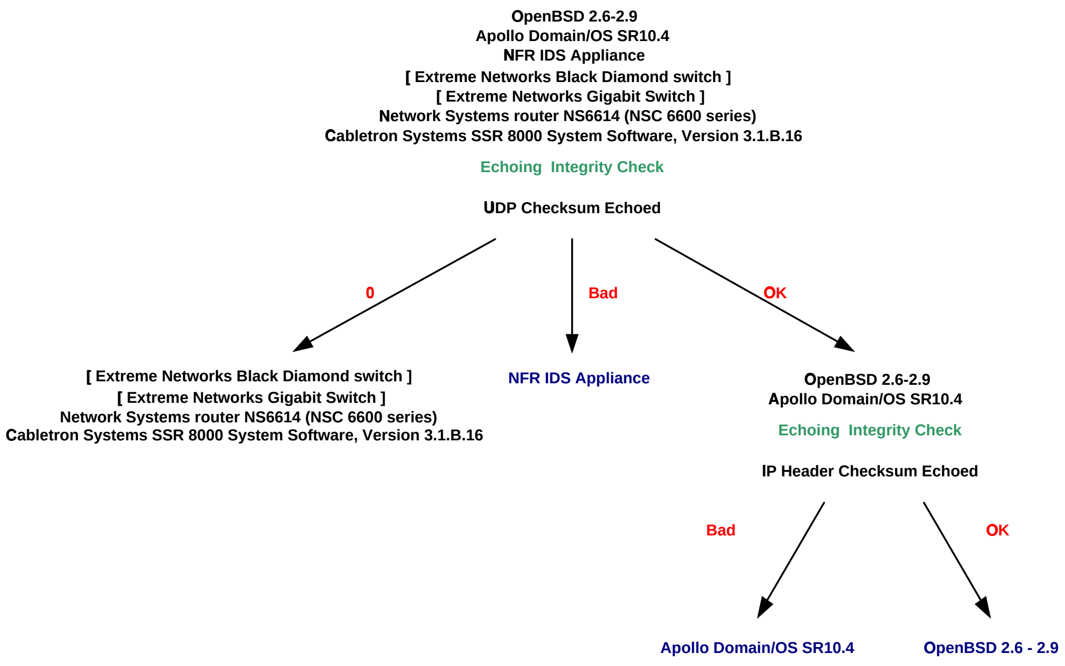
OpenBSD 2.6-2.9
Apollo Domain/OS SR10.4
NFR IDS Appliance
[Extreme Networks Black Diamond switch]
[Extreme Networks Gigabit Switch]
Network Systems router NS6614 (NSC 6600 series)
Cabletron Systems SSR 8000 System Software, Version 3.1.B.16
Echoing Integrity Check
UDP Checksum Echoed
0
Bad
OK
[Extreme Networks Black Diamond switch]
[Extreme Networks Gigabit Switch]
Network Systems router NS6614 (NSC 6600 series)
Cabletron Systems SSR 8000 System Software, Version 3.1.B.16
NFR IDS Appliance
OpenBSD 2.6-2.9
Apollo Domain/OS SR10.4
Echoing Integrity Check
IP Header Checksum Echoed
Bad
OK
Apollo Domain/OS SR10.4
OpenBSD 2.6 - 2.9

Figure 8: Dividing the IP Total Length echoed $<$ 20 bytes group

The first echoing integrity test will be with the IP Header checksum being echoed. While AIX will miscalculate the value being echoed, the other operating systems will zero out this field value.

The second echoing integrity test will be with the IP Identification field value being echoed. Because of Bit ordering problems BSDI 2.x, 3.x, and NetBSD 1.x-1.2.x little Endian will not echo correctly the IP Identification field value, while BSDI 4.x, NetBSD 1.x-1.2.x big Endian and MacOS X 1.0-1.2 will echo it correctly.

The ICMP fingerprinting methods allow us to group together several operating systems with the BSDI/NetBSD 1.x-1.2.x/MacOS X 1.0-1.2 branch.

If you will examine closely these two groups you will understand they are all based on the same TCP/IP base code.

Continuing with the main branch (IP Total Length field value echoed correctly)

The next fingerprinting method to be used is another member of the Echoing

Error Message and not the one with the offending packet's echoed data

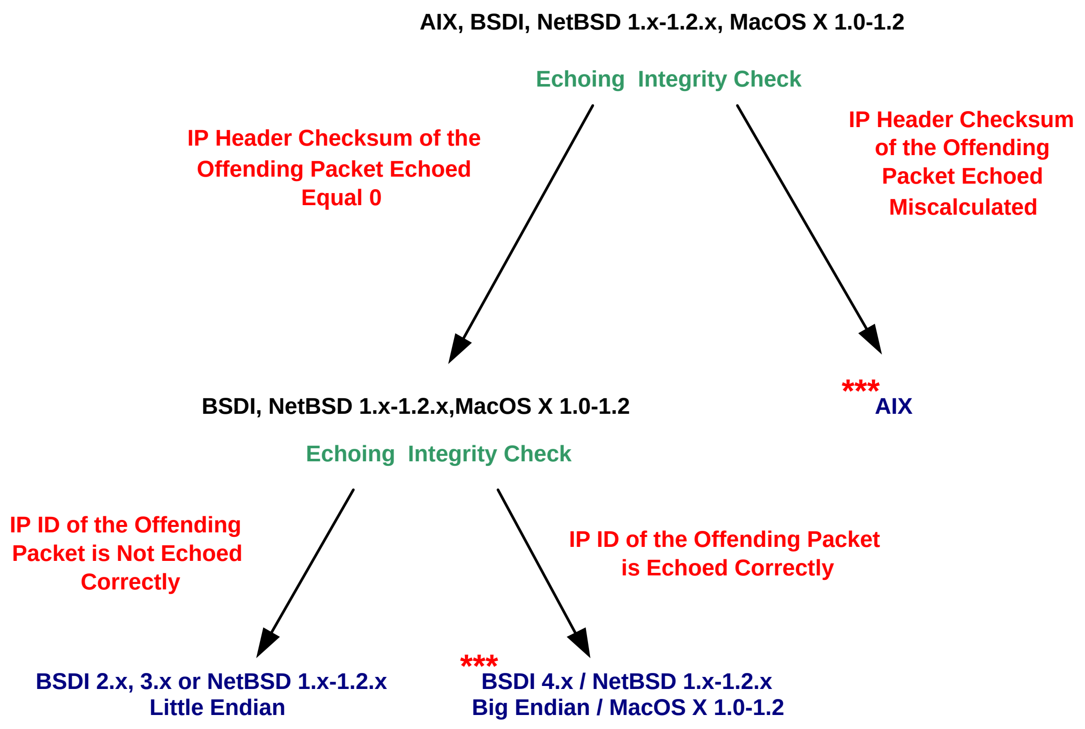
AIX, BSDI, NetBSD 1.x-1.2.x, MacOS X 1.0-1.2
Echoing Integrity Check
IP Header Checksum of the Offending Packet Echoed Equal 0
IP Header Checksum of the Offending Packet Echoed Miscalculated
BSDI, NetBSD 1.x-1.2.x,MacOS X 1.0-1.2
*** AIX
Echoing Integrity Check
IP ID of the Offending Packet is Not Echoed Correctly
IP ID of the Offending Packet is Echoed Correctly
BSDI 2.x, 3.x or NetBSD 1.x-1.2.x Little Endian
*** BSDI 4.x / NetBSD 1.x-1.2.x Big Endian / MacOS X 1.0-1.2

Figure 9: Dividing the IP Total Length echoed > 20 bytes group

Integrity family.

This time we are examining the 3bits (Unused, MF, DF) flags and offset fields. Several operating systems, when a value is given with this fields with an offending packet, will change the bit ordering with their ICMP error messages.

The next example is with NetBSD 1.3:

```
21:46:07.489298 eth0 > 172.18.2.201.1144 > 172.18.2.20.re-mail-ck:
                                      udp 80 (DF) [tos 0x11] (ttl 64, id 44586)

4511 006c ae2a 4000 4011 2f44 ac12 02c9
ac12 0214 0478 0032 0058 cfc4 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
```

```
21:46:07.489298 eth0 < 172.18.2.20 > 172.18.2.201:
               icmp: 172.18.2.20 udp port re-mail-ck unreachable Offending pkt:
172.18.2.201 > 172.18.2.20: (frag 10926:88@512)
                        [tos 0x11] (ttl 64, bad cksum 0!) (DF) (ttl 255, id 56)

4500 0038 0038 4000 ff01 1e8b ac12 0214
ac12 02c9 0303 ea7b 0000 0000 4511 006c
2aae 0040 4011 0000 ac12 02c9 ac12 0214
0478 0032 0058 0000
```

Looking closely at the tcpdump trace above, we can see that the DF bit is set with the offending UDP datagram. Looking at the ICMP Port Unreachable error message at the echoed data part, the Bit order has changed from 4000 to 0040. This made the offending packet look like a fragmented datagram that tried to access a closed UDP port.

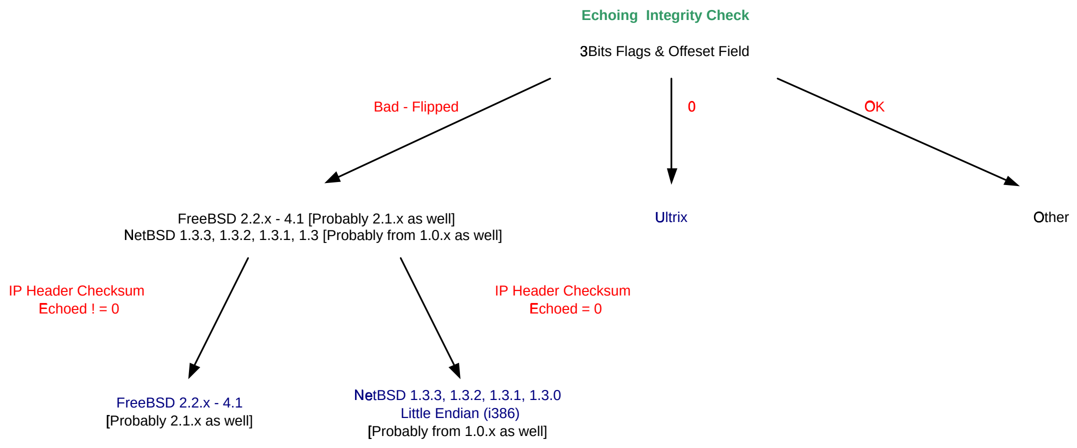
Echoing Integrity Check
3Bits Flags & Offeset Field
Bad - Flipped
0
OK
FreeBSD 2.2.x - 4.1 [Probably 2.1.x as well]
NetBSD 1.3.3, 1.3.2, 1.3.1, 1.3 [Probably from 1.0.x as well]
Ultrix
Other
IP Header Checksum
Echoed ! = 0
IP Header Checksum
Echoed = 0
FreeBSD 2.2.x - 4.1
[Probably 2.1.x as well]
NetBSD 1.3.3, 1.3.2, 1.3.1, 1.3.0
Little Endian (i386)
[Probably from 1.0.x as well]

Figure 10: 3Bits Flags & Offset Fields

Using this Echoing Integrity variation we are able to isolate FreeBSD 2.2.x-4.1 based operating systems and NetBSD 1.3.x based operating systems.

Using another Echoing Integrity check, which examines the IP Header checksum being echoed we are able to divide between FreeBSD 2.2.x-4.1 based machines (IP Header checksum is not zero) and the NetBSD 1.3.x based machines (IP Header Checksum 0)

Please note that we might see similar behavior with older versions of NetBSD

(1.x-1.2.x), and with older versions of BSDI.

While FreeBSD 2.2.x-4.1 and NetBSD 1.3.x based machines will change the Bit order with the value being echoed with the ICMP error message, Ultrix based machines will send zero (0) as the field value with the echoed IP Header with the ICMP error message.

We are again at the main branch.

Identifying Microsoft Based Operating Systems

In order to isolate the Microsoft based operating systems from the rest of the questionable IP addresses left, we need to initiate another query.

We will use a method that sends code field value different than zero (0) with ICMP Echo requests. The Microsoft based operating systems will use zero (0) for their code field value with the ICMP echo replies they will produce, while the rest of the OS world will use the same value given with the ICMP echo request (as the RFC states).

With our ICMP echo request message we will be using another fingerprinting method called "TOS Echoing". We will set the TOS field to a certain field value with our ICMP echo request. Only certain operating systems will not echo back this field value with their ICMP echo replies.

We will also set the DF bit with the request. It will enable us to use the "DF Bit Echoing with ICMP Queries" fingerprinting method.

If we will not receive a reply for our ICMP echo request, than ICMP echo requests are being filtered.

The MS based Operating Systems branch

Using the IP Time-to-Live field value we are able to identify Microsoft Windows 95 based IP addresses. They will use a starting value of 32 for their IP Time-to-Live field while the other Microsoft based operating systems will use 128.

The next criteria to be used is the "TOS Echoing". Since Microsoft Windows 2000 family of operating systems will not echo the TOS field value with their ICMP Echo replies, we can isolate this group of operating systems from the rest

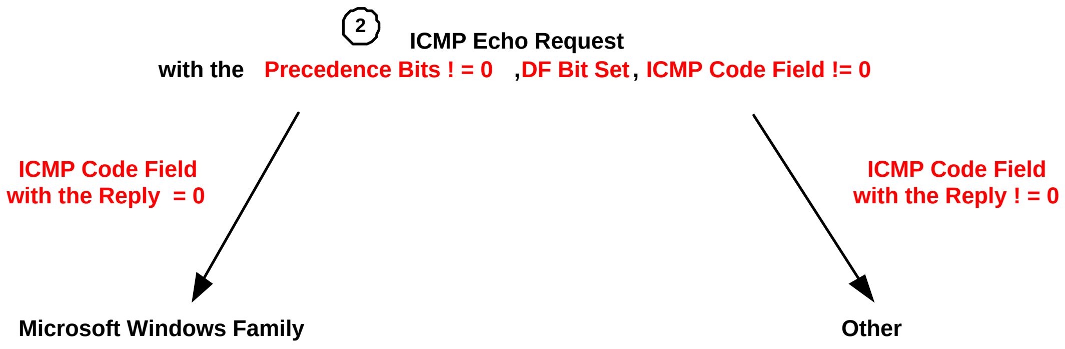
2
ICMP Echo Request
with the Precedence Bits ! = 0 ,DF Bit Set, ICMP Code Field != 0
ICMP Code Field with the Reply = 0
ICMP Code Field with the Reply ! = 0
Microsoft Windows Family
Other

Figure 11: Identifying MS based OSs

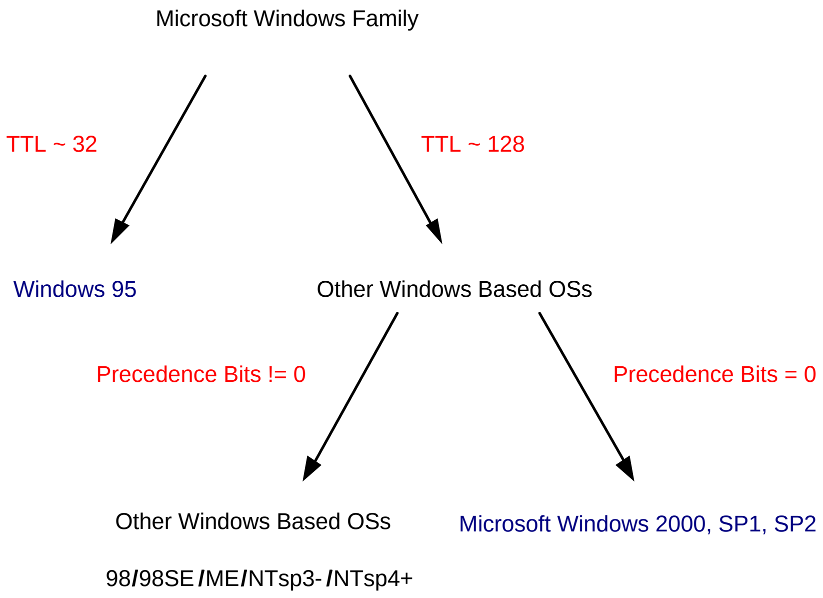

Figure 12: Identifying MS based OSs

of the Microsoft based operating systems.

Using the "Who Answer What?" approach we will divide the rest of the Microsoft based operating systems, as the diagram below suggests.

Back to the main branch

From the operating systems left, Ultrix and Novell are the only operating systems that will not echo back the DF bit with their ICMP Echo replies.

Using the IP Time-to-Live field value we will be able to differentiate between the two.

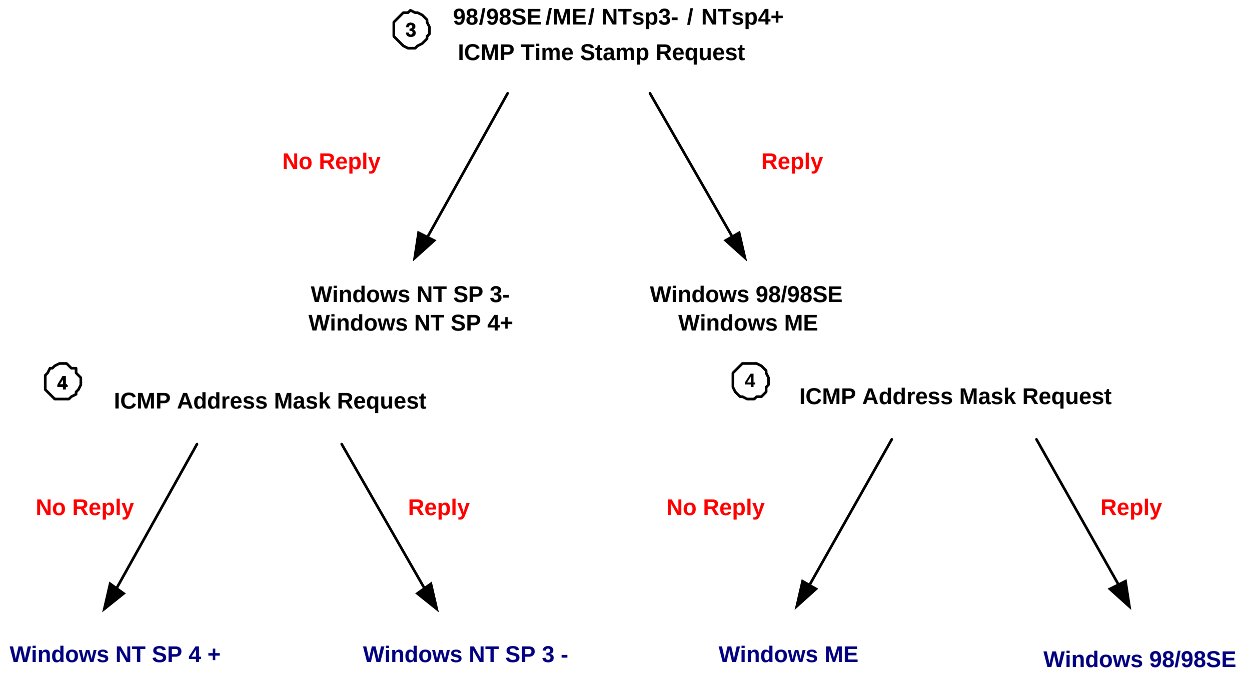

Figure 13: Identifying MS based OSs

DF Bit Echoing With ICMP Error Messages

Looking again at the ICMP Error message triggered by the offending UDP datagram we have sent, we will examine another fingerprinting technique - ”DF Bit echoing with ICMP Error messages”. Several operating systems will not set the DF bit with the IP Header of the ICMP Error message, although we have set it with our offending UDP datagram.

From the list of operating systems left, OpenBSD 2.1-2.5, and NetBSD 1.4, 1.5 will produce this kind of behavior.

Since OpenBSD 2.1-2.3 based machines will zero out the UDP checksum with their ICMP port unreachable error messages, we can divide this group further.

Main Branch - What is still left?

Until this point of the decision tree, the results we will receive are very reliable.

From this point, it is very important to identify the target’s location/topology.

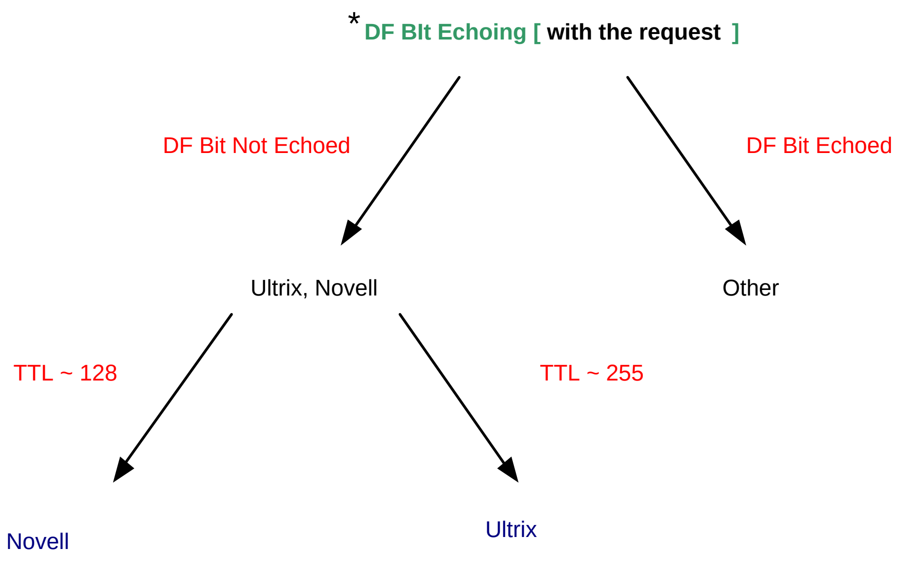

Figure 14: Using DF Bit Echoing with ICMP Query Messages

- Are we inside an Intranet?
- Are we on the Internet?
- Are we querying a host on another Intranet subnet?
- Are we querying a host on the Internet?

The next test is using the "Who answer what?" approach. The problem is that my mapping of networking devices is far from being complete, and I cannot conclude which networking devices will answer/not answer an ICMP information request, for example.

This is another reason why I have stated that it is very important to understand the surroundings of the target we are working against.

We will look for operating systems that answer ICMP Information request. The answering

IP addresses will be checked for echoing integrity problems with their ICMP error message they produced in step one of the decision tree.

The IP addresses that will answer Information request messages will be OpenVMS, HPUX 10.20, and DGUX.

What's next?

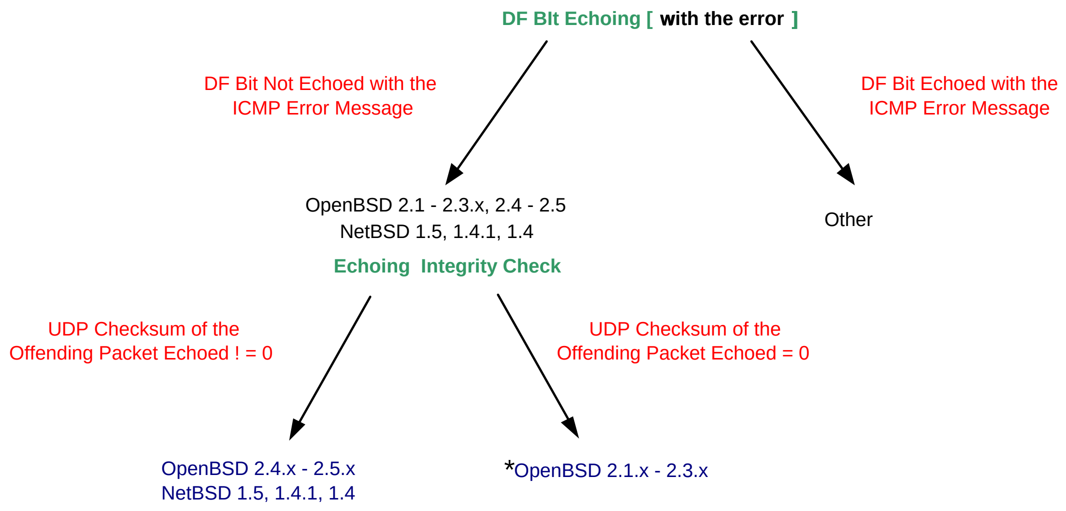
DF BIt Echoing [with the error]
DF Bit Not Echoed with the ICMP Error Message
DF Bit Echoed with the ICMP Error Message
OpenBSD 2.1 - 2.3.x, 2.4 - 2.5
NetBSD 1.5, 1.4.1, 1.4
Echoing Integrity Check
Other
UDP Checksum of the Offending Packet Echoed ! = 0
UDP Checksum of the Offending Packet Echoed = 0
OpenBSD 2.4.x - 2.5.x
NetBSD 1.5, 1.4.1, 1.4
*OpenBSD 2.1.x - 2.3.x

Figure 15: Using DF Bit Echoing with ICMP Error Messages

After this stage the quality of results is questionable.

I have decided to stop at this point at the moment, although continuing from here and identifying, for example, FreeBSD 4.1.1, 4.2, 4.3 and 5.0 beta is very simple.

4 The Future development of X and xprobe

X goal is to be efficient and intelligent in determining the target IP stack. This means that other features and capabilities will be added, and should be added.

Mainly two OS fingerprinting methods are planned to be used: hardcoded "AI" mechanism (currently implemented) and singnature-based approach.

We are planning on adding some custom scenarios where one can specify the topology which Xprobe should be working against. For example, if Xprobe is being used on an internal LAN it may require one set of tests, where when used to probe another host on the Internet it may require another set of tests.

We are planning on adding some fail over mechanisms to X. This means that if a certain test will fail because we suspect a filtering device is blocking our queries we will be able to 'fall' into another scenario and logic. It will not only be implemented if the ICMP Port Unreachable Error Message will not be

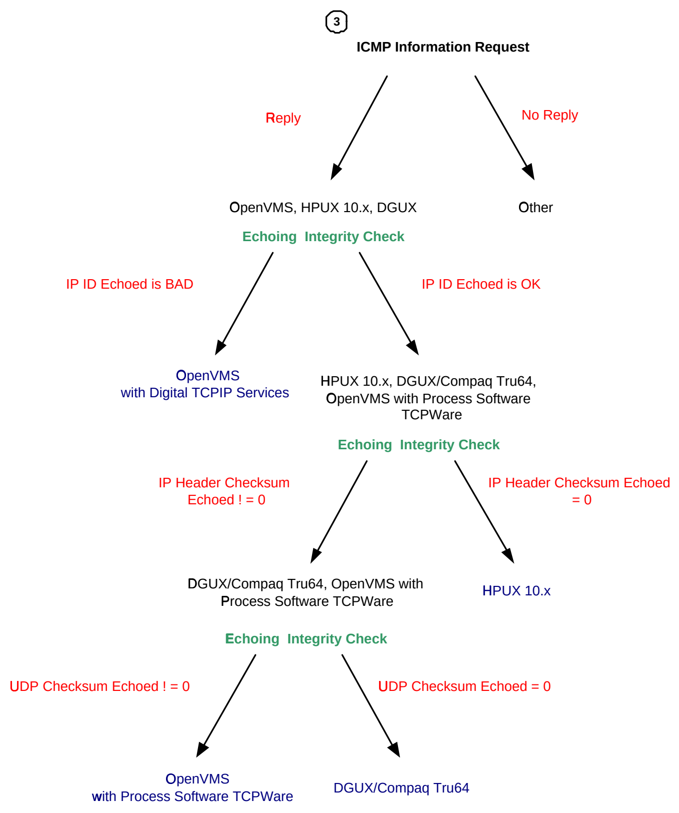
3
ICMP Information Request
Reply
No Reply
OpenVMS, HPUX 10.x, DGUX
Other
Echoing Integrity Check
IP ID Echoed is BAD
IP ID Echoed is OK
OpenVMS with Digital TCPIP Services
HPUX 10.x, DGUX/Compaq Tru64, OpenVMS with Process Software TCPWare
Echoing Integrity Check
IP Header Checksum Echoed ! = 0
IP Header Checksum Echoed = 0
DGUX/Compaq Tru64, OpenVMS with Process Software TCPWare
HPUX 10.x
Echoing Integrity Check
UDP Checksum Echoed ! = 0
UDP Checksum Echoed = 0
OpenVMS with Process Software TCPWare
DGUX/Compaq Tru64

Figure 16: Using ICMP Information Requests and Replies

received from a targeted machine, but also if ICMP Echo replies will not be received, and might be implemented for other scenarios as well.

Also elements of network mapping will be included (ttl distance precalculation, optional reachability scanning, optional closed port search mechanism etc).

We know we have some limitation with Xprobe:

- Xprobe is based on a static logic (X)
- Xprobe can identify only Operating Systems and a small number of networking devices
- Xprobe is ICMP/UDP based (i.g. if these protocols are filtered we fail)

We are planning some enhancements:

- Relaying on a fingerprinting Database.

- Using a dynamic logic with an internal 'AI'.
- Adding vast support for networking devices.
- Implementing different tests according to different topologies.
- Implementing several filtering devices identification tests.
- Using real application databases with the UDP query.
- Certain network mapping capabilities shall be included (ttl distance pre-calculation, search for closed UDP port, reachability tests, etc).

A Appendix: Summary of tests

Please refer to the table 1 for details.

Operating System	Queries to ID	Techniques Combo Used
Linux Kernel 2.0.x	1	Precedence Bits Echoing, Amount of Data Echoed, IP TTL
Cisco Routers based IOS 11.x-12.0.x	1	Precedence Bits Echoing, Amount of Data Echoed, UDP Checksum Echoed
Extreme Network Switches	1	Precedence Bits Echoing, Amount of Data Echoed, UDP Checksum Echoed
Linux Kernel 2.2.x / 2.4.1-2.4.5	2	Precedence Bits Echoing, Amount of Data Echoed, IP TTL, IPID
Linux Kernel 2.4.0	2	Precedence Bits Echoing, Amount of Data Echoed, IP TTL, IPID
MacOS 7.x - 9.x	2	Precedence Bits Echoing, Amount of Data Echoed, Unused Bit
Sun Solaris 2.3-2.8	3	Precedence Bits Echoing, Amount of Data Echoed, Echo request with Unused Bit set, Timestamp request
HPUX 11.x	3	Precedence Bits Echoing, Amount of Data Echoed, Echo request with Unused Bit set, Timestamp request
AIX 3.x, 4.x	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, IP Header Checksum Echoed

Operating System	Queries to ID	Techniques Combo Used
BSDI 2.x, 3.x ; NetBSD 1.x-1.2.x little Endian	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, IP Header Checksum Echoed, IP ID field value echoed
BSDI 4.x; NetBSD 1.x-1.2.x big Endian; MacOS X 1.0-1.2	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, IP Header Checksum Echoed, IP ID field value echoed
FreeBSD 2.2.x-4.1	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, IP Header Checksum
NetBSD 1.3.x	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, IP Header Checksum
Ultrix	1	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields
MS Windows 95	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field
MS Windows 2000	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, TOS field value

Operating System	Queries to ID	Techniques Combo Used
Microsoft Windows 98/98SE	4	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, TOS field value, ICMP Timestamp request, ICMP Address Mask request
Microsoft Windows ME	4	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, TOS field value, ICMP Timestamp request, ICMP Address Mask request
Microsoft Windows NT4 SP 3 and below	4	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, TOS field value, ICMP Timestamp request, ICMP Address Mask request
Microsoft Windows NT4 SP4 and above	4	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, TOS field value, ICMP Timestamp request, ICMP Address Mask request

Operating System	Queries to ID	Techniques Combo Used
Ultrix	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing, IP TTL
Novell	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing, IP TTL
OpenBSD 2.1-2.3	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages
OpenBSD 2.4-2.5 / NetBSD 1.4, 1.5	2	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages
OpenVMS with Digital TCP/IP Services	3	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages, ICMP Information request, IPID echoing integrity

Operating System	Queries to ID	Techniques Combo Used
HPUX 10.20	3	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages, ICMP Information request, IPID echoing integrity, IP Header Checksum
DGUX / Compaq Tru64	3	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages, ICMP Information request, IPID echoing integrity, IP Header Checksum, UDP Checksum
OpenVMS with Process software TCPware	3	Precedence Bits Echoing, Amount of Data Echoed, IP Total Length Field Value Echoed, 3Bits Flags and Offset fields, ICMP Code field, DF Bit Echoing with Error Messages, ICMP Information request, IPID echoing integrity, IP Header Checksum, UDP Checksum

Table 1: Summary of Tests

B Appendix: Availability

The developed software, and relevant documentation is available at following locations:

http://www.sys-security.com/html/projects/X.html

http://xprobe.sourceforge.net

http://www.notlsd.net/xprobe/

References

[1] http://www.sys-security.com

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

[3] RFC 791 - Internet Protocol `http://www.ietf.org/rfc/rfc791.txt`

[4] RFC 1812 - Requirements for IP Version 4 Routers `http://www.ietf.org/rfc/rfc1812.txt`

[5] RFC 792 - The ICMP Protocol `http://www.ietf.org/rfc/rfc792.txt`

[6] RFC 1349 - Type of Service in the Internet Protocol Suite `http://www.ietf.org/rfc/rfc1349.txt`