Xprobe

Remote ICMP Based OS Fingerprinting Techniques

Ofir Arkin

Managing Security Architect

Where Security & Business Intersect℠

Agenda

- What is Xprobe?
- Xprobe 101
- Examples
- The Static Engine
- The Signature Based Engine
- More Examples
- Known Problems, Detecting Xprobe's Activity, To Do List
- Questions

Xprobe Creators

What is Xprobe?

Written and maintained by Fyodor Yarochkin and Ofir Arkin, Xprobe is an Active OS fingerprinting tool based on Ofir Arkin's ICMP Usage In Scanning Research project (http://www.sys-security.com).

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 XP & 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.

What is Xprobe?

As we will demonstrate the number of datagrams we need to send and receive in order to remotely fingerprint a targeted machine with Xprobe 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 amount of packets used to successfully identify an operating system is maximum of 4 sent, and maximum of 4 received.

…This makes Xprobe very fast as well.

What is Xprobe?

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 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, and make Xprobe transparent to a lot of IDS systems.

What is Xprobe?

Xprobe 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 Xprobe we combine the host detection stage with the operating system detection stage. With maximum of four datagrams initiated from the prober's machine, we are able to determine if a certain machine is running an operating system where certain vulnerabilities might be presented.

What is Xprobe?

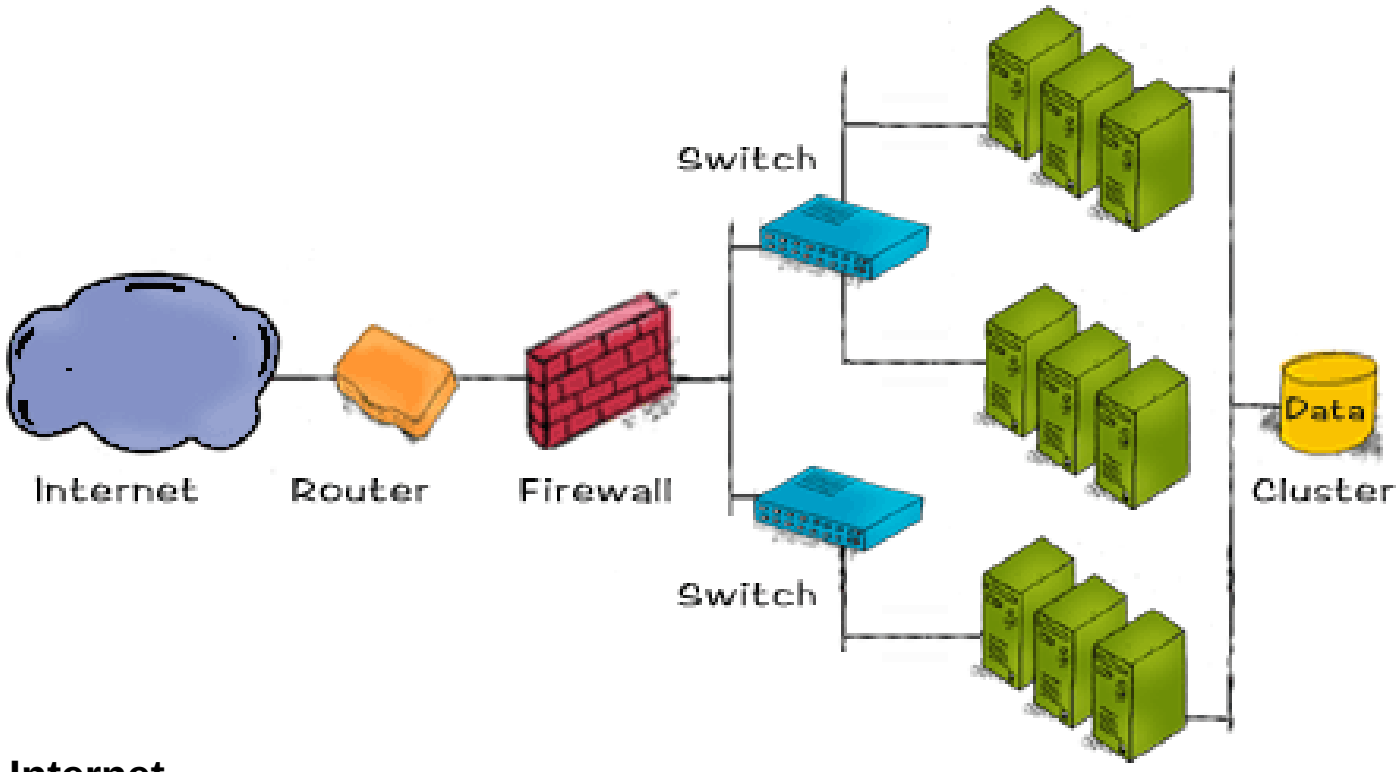
For example, a Microsoft Windows 2000 based operating system (and Microsoft Windows XP) can be identified with four datagrams traversing over the network in total (two sent and two received).

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

Xprobe Introduction

- First Introduced at the Black Hat Briefings July 2001 Las – Vegas, USA (v0.0.1). Current versions are 0.0.2 and 0.1.
- The logic behind the tool is called X.
- Compiles on Linux Kernel 2.0.x, 2.2,x, and 2.4.x series, *BSD, Sun Solaris, & IRIX.
- The tool is Fast, Efficient, Small, and Simple.
- Xprobe has 2 development trees: 0.0.x for Static decision Tree (limited), and 0.1.x for Signature Dynamic DB support.
- In the future the static side of Xprobe will be combined from different static logics (depending on the topology) where a *smart* dynamic *fail-over* mechanism is to lunch one of the logics if the start parameters of the first logic fails.
- Xprobe version 0.1 works against a signature database. We are looking to add dynamic solution logic as well.

Topology Matters



- **Internet**
- **Local LAN**
- **Between LAN Segments**

Xprobe License

Xprobe Compilation

tar xvfz xprobe-{release}.tar.gz

cd xprobe-{release}

./configure

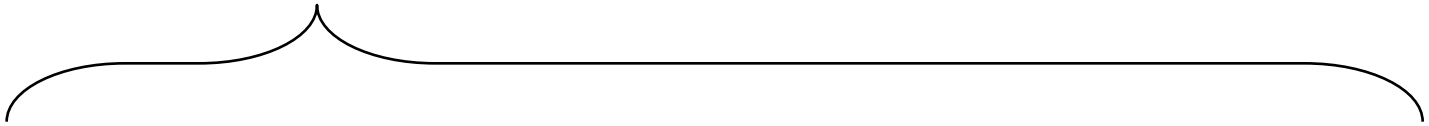
(or ./configure --with-libpcap-libraries=/usr/local/lib --with-libcap-includes=/usr/local/include)

make

make install

Xprobe Usage

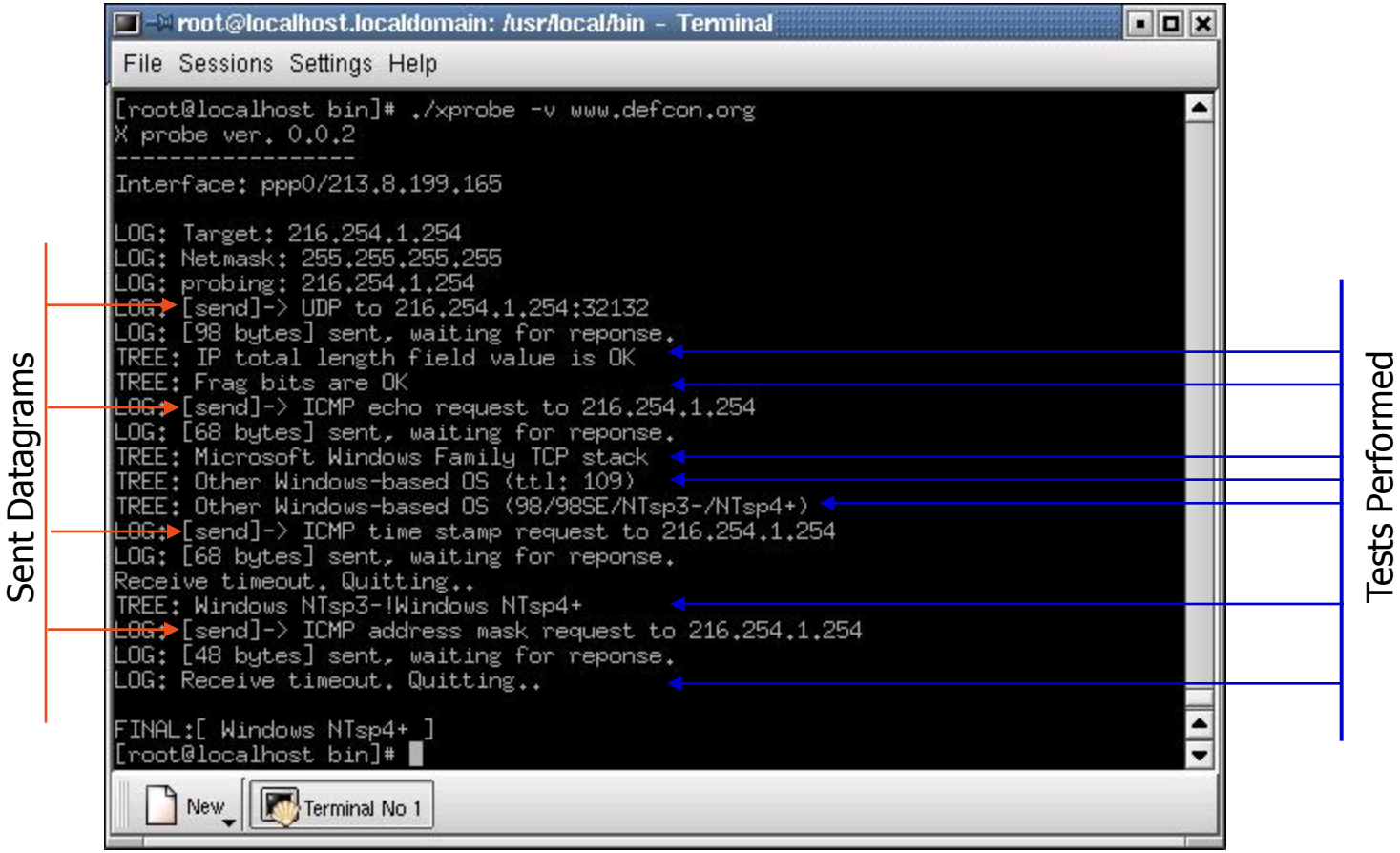
xprobe [options] hostname[/netmask]



```
-h help
-v be verbose
-i <interface> run on interface
-p <portnum>   use <portnum> udp port for udp probe
-o logfile     log everything into a logfile
```

Example: www.defcon.org

Example: www.defcon.org



Example: www.defcon.org

Start Time

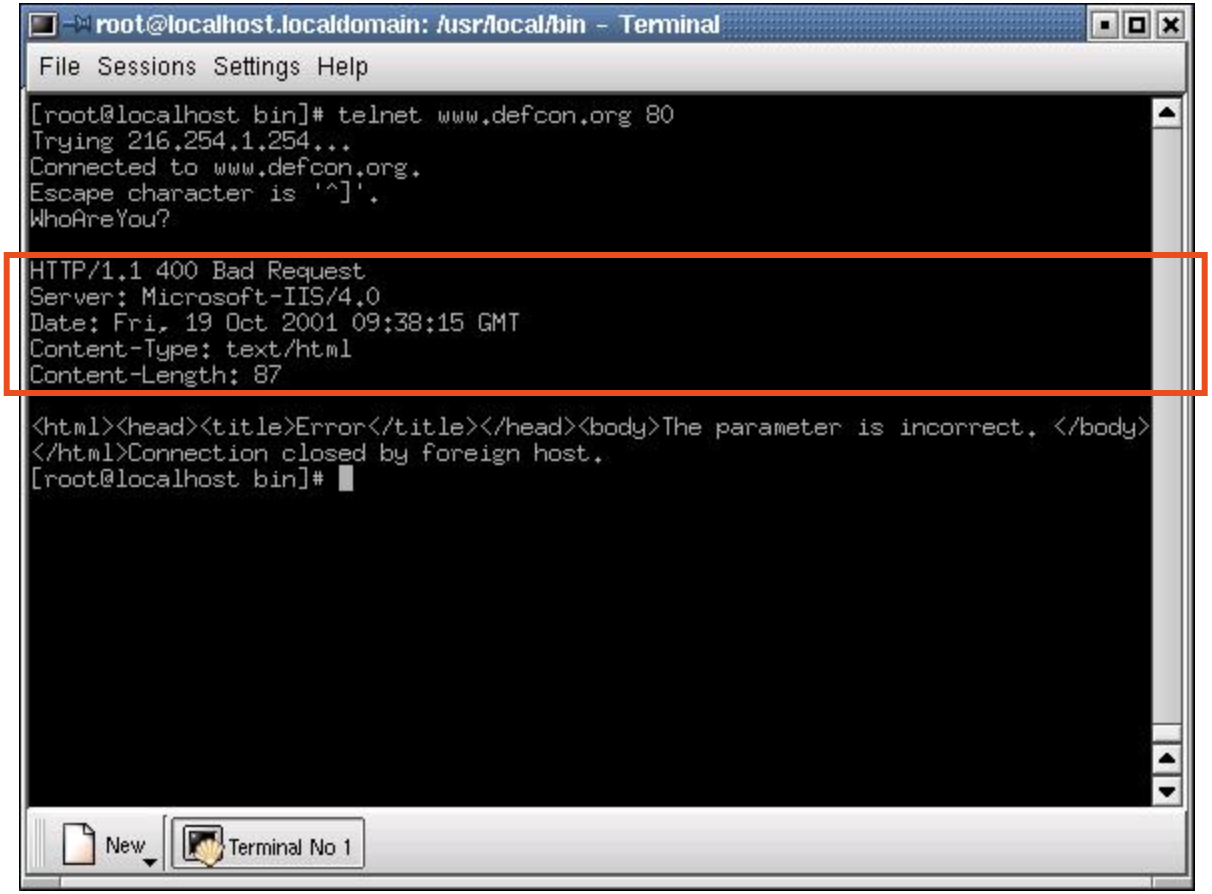
```
10/19-19:31:00.791716 213.8.199.165:32426 -> 216.254.1.254:32132
UDP TTL:250 TOS:0x0 ID:47464 IpLen:20 DgmLen:98 DF
Len: 78
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00                                ......


10/19-19:31:01.161716 216.254.1.254 -> 213.8.199.165
ICMP TTL:109 TOS:0x0 ID:4224 IpLen:20 DgmLen:56
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
213.8.199.165:32426 -> 216.254.1.254:32132
UDP TTL:232 TOS:0x0 ID:47464 IpLen:20 DgmLen:98
Len: 78
** END OF DUMP
00 00 00 00 45 00 00 62 B9 68 40 00 E8 11 61 77  ....E..b.h@...aw
D5 08 C7 A5 D8 FE 01 FE 7E AA 7D 84 00 4E 8B 78  ........~.}..N.x
```

Example: www.defcon.org

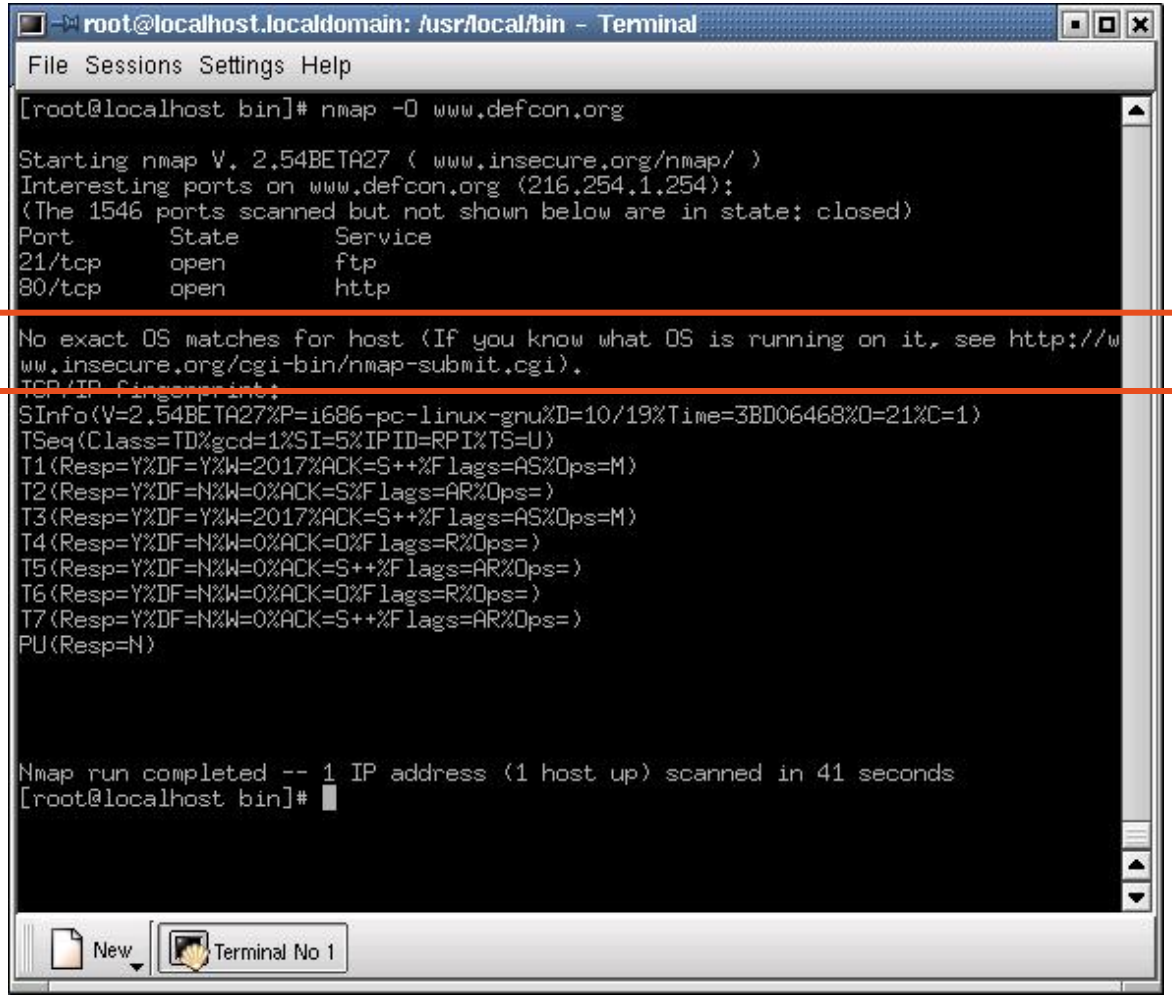
```
10/19-19:31:01.161716 213.8.199.165 -> 216.254.1.254
ICMP TTL:250 TOS:0x6 ID:25934 IpLen:20 DgmLen:68 DF
Type:8  Code:123  ID:10421   Seq:30396  ECHO
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00                          ........

10/19-19:31:01.531716 216.254.1.254 -> 213.8.199.165
ICMP TTL:109 TOS:0x2 ID:4480 IpLen:20 DgmLen:68 DF
Type:0  Code:0  ID:10421  Seq:30396  ECHO REPLY
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00                          ........

10/19-19:31:01.531716 213.8.199.165 -> 216.254.1.254
ICMP TTL:250 TOS:0x0 ID:25006 IpLen:20 DgmLen:68
Type:13  Code:0  TIMESTAMP REQUEST
70 FC C6 DD 00 00 00 00 00 00 00 00 00 00 00 00  p...............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00              ............

10/19-19:31:11.161716 213.8.199.165 -> 216.254.1.254
ICMP TTL:250 TOS:0x0 ID:48938 IpLen:20 DgmLen:48
Type:17  Code:0  ADDRESS REQUEST
00 00 8E 5A 00 00 00 00 00 00 00 00 00 00 00 00  ...Z............
00 00 00 00 00 00 00 00                          ........
```

Finish Time ~380ms [+ 250ms waiting time for the ICMP Address Mask Reply]

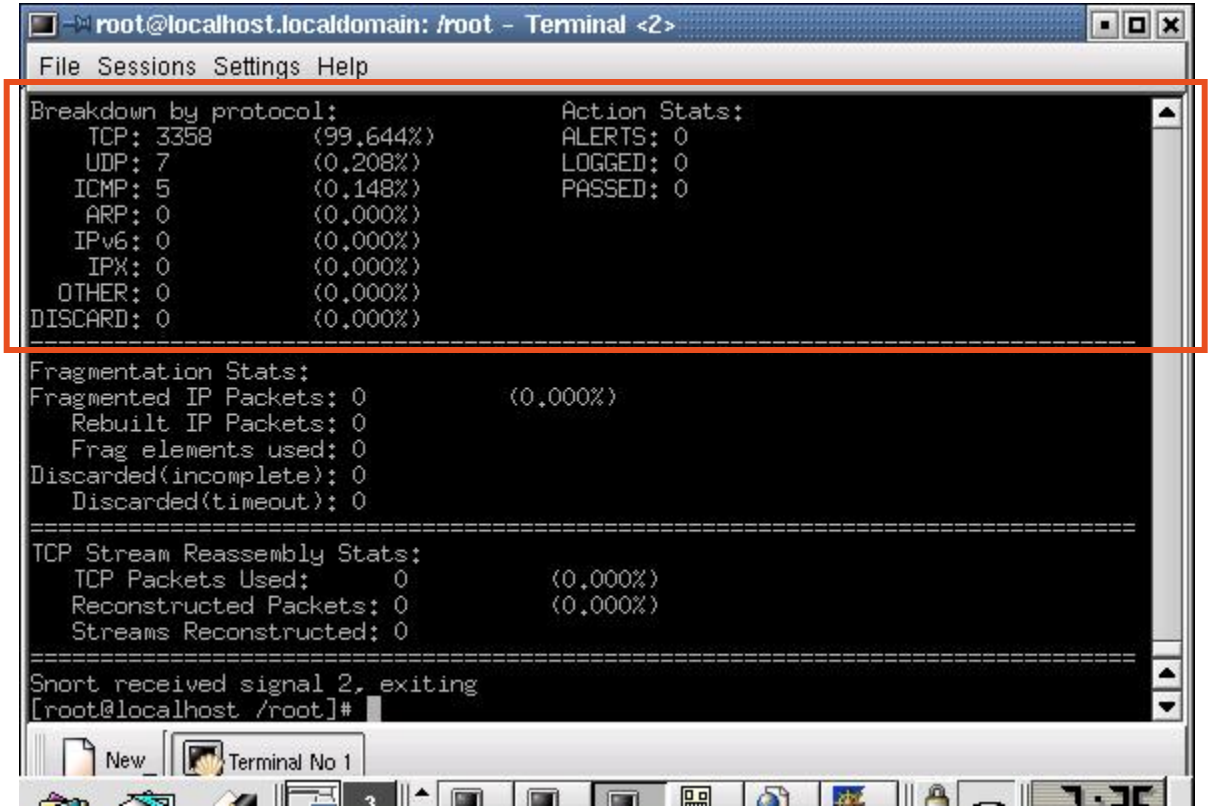
Example: www.defcon.org



Example: www.defcon.org



Example: www.defcon.org



Why Xprobe? - Pros

- The ICMP protocol hasn't been looked onto from the point of remote OS fingerprinting. Each 'vendor' implemented it in quite relaxed manner, with 'deviations' which are continuously being reproduced in every release of TCP/IP stack from certain vendor and at times more reliable to be reproduced than TCP 'tests'.
- Using some kind of "AI" (or an analysis) for a scanning tool is a lot smarter than just choking the network with huge amount of packets.
- Small overhead in the Scanning process
- No sudden Denial-of-Service or other 'surprises' during the scan.
- Fast - Active OS fingerprinting using the ICMP protocol uses small amount of packets sent and received.
- Stealth - People think they were 'pinged' were they actually been mapped.
- Accurate - Today we are using tools that are, sometimes, *inaccurate* and *inconsistent* with their results. Version 0.1 makes Xprobe even more accurate.
- Able to differentiate between any Microsoft Windows based TCP/IP Stack

Why Xprobe? - Cons

- Limited to the ICMP and UDP protocols only
- Internet usage can be defeated (smart firewall rule base)
- Failover to TCP, or a correlation with TCP needs to be done, in a smart manner. This will lead to a very accurate Active OS fingerprinting tool using very small amount of traffic to determine a remote OS.
- The Static Decision Tree is limited in adding new operating systems and networking devices (the static decision tree is hardcoded into the binary).

Xprobe – Inner Working

- Xprobe has 2 development trees: 0.0.x for Static decision Tree (limited in the number of operating systems and networking devices it supports), and 0.1.x for Signature Dynamic DB support.
- The difference is with the processing of the results from the tests.
  - With the 0.0.x dev-tree we have a predefined static decision tree. Adding operating systems and networking devices is a hard task since it needs to be hard coded.
  - With the 0.x dev-tree we use a signature database support. This means that in order to add support for an operating system or a networking device all we need is create an entry in the signature database.
  - In a sentence: Version 0.1.x - more accurate, easier and more flexible way to maintain and update signatures.
- Both development trees use the same Active OS fingerprinting methods using the ICMP protocol.

Xprobe - 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.

Xprobe - ICMP Error Message Echoing Integrity

With Xprobe 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

Xprobe - ICMP Error Message Echoing Integrity

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.

Xprobe - ICMP Error Message Echoing Integrity

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.

Xprobe - ICMP Error Message Echoing Integrity

An example with AIX 3.2

```
15:44:56.822182 ppp0 > 32.101.233.50.36196 > y.y.y.y.32132: udp 70 (DF)
(ttl 250, id 2279)
                        4500 0062 08e7 4000 fa11 d99b 2065 e932
                        yyyy yyyy 8d64 7d84 004e 5661 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000
```

IP Total Length Field Value Echoed is 118 while the Original was 98

```
15:44:57.192182 ppp0 < y.y.y.y > 32.101.233.50: icmp: y.y.y.y udp port
32132 unreachable Offending pkt: 32.101.233.50.36196 > y.y.y.y.32132:
udp 70
(DF) (ttl 234, id 2279, bad cksum e99b!) (DF) (ttl 233, id 40032)
                        4500 0038 9c60 4000 e901 575c yyyy yyyy
                        2065 e932 0303 f1b1 0000 0000 4500 0076
                        08e7 4000 ea11 e99b 2065 e932 yyyy yyyy
                        8d64 7d84 004e 0000
```

IP Header Checksum Echoed is Miscalculated

Xprobe - ICMP Error Message Echoing Integrity

An example with OpenBSD 2.8

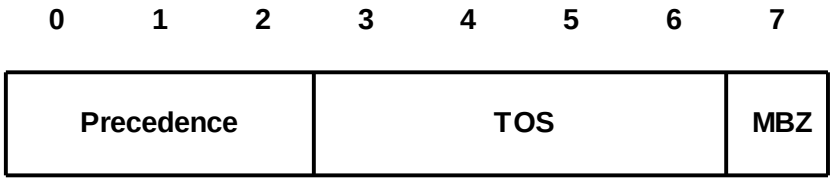
```
15:47:25.342182 ppp0 > 32.101.233.50.53783 > y.y.y.y.32132: udp 70 (DF)
(ttl 250, id 57568)
                        4500 0062 e0e0 4000 fa11 1dad 2065 e932
                        yyyy yyyy d217 7d84 004e 2db9 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000
```

IP Total Length Field Value Echoed is 20 Bytes less than the Original

```
15:47:25.652182 ppp0 < y.y.y.y > 32.101.233.50: icmp: y.y.y.y udp port
32132 unreachable Offending pkt: 32.101.233.50.53783 > y.y.y.y.32132:
udp 70 (DF) (ttl 238, id 57568) (ttl 241, id 61090)
                        4500 0038 eea2 0000 f101 5925 yyyy yyyy
                        2065 e932 0303 7f59 0000 0000 4500 004e
                        e0e0 4000 ee11 29c1 2065 e932 yyyy yyyy
                        d217 7d84 004e 2db9
```

Xprobe - Precedence Bits Issues

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", which is 3-bit long, is intended to prioritize the IP Datagram. It has eight levels of prioritization.

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.

Xprobe - Precedence Bits Issues

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".

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

Xprobe - 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.

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, Nokia FW boxes (and other OSs and several Networking Devices) are good examples.

Xprobe - ICMP Error Message Quoting Size

An example with Linux Kernel 2.4.6

```
15:47:47.729742 ppp0 > x.x.x.x.47612 > y.y.y.y.32132: udp 70 (DF) (ttl 250, id 121)
                       4500 0062 0079 4000 fa11 c32d xxxx xxxx
                       yyyy yyyy b9fc 7d84 004e 0aed 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000
15:47:47.889742 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 32132 unreachable
Offending pkt: x.x.x.x.47612 > y.y.y.y.32132: udp 70 (DF) (ttl 242, id 121) [tos
0xc0]  (ttl 245, id 45284)
                       45c0 007e b0e4 0000 f501 56f6 yyyy yyyy
                       xxxx xxxx 0303 ba40 0000 0000 4500 0062
                       0079 4000 f211 cb2d xxxx xxxx yyyy yyyy
                       b9fc 7d84 004e 0aed 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000 0000
                       0000 0000 0000 0000 0000 0000 0000
```

(1) Precedence Field Vale is 0xc0

(2) Data Echoed

UDP Header Information

Xprobe - 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 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.

Xprobe - 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.
- 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.

Xprobe - TOS Echoing

An example with Microsoft Windows 2000

```
17:13:13.081831 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request (DF)
[tos 0x6,ECT]  (ttl 250, id 2779)
                        4506 0044 0adb 4000 fa01 a9a9 xxxx xxxx
                        yyyy yyyy 087b 2fff a51e 2267 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000
17:13:13.231831 ppp0 < y.y.y.y > x.x.x.x.175: icmp: echo reply (DF) (ttl
115, id 59514)
                        4500 0044 e87a 4000 7301 5310 yyyy yyyy
                        xxxx xxxx 0000 387a a51e 2267 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000
```

Code Field = 0

Precedence Bits Value = 0

Xprobe - The Rest

DF Bit

Will the DF Bit will be set with a reply ICMP message of any kind?

IP Time-To-Live

IP Time-To-Live Field Value with both ICMP Echo Requests (and ICMP Error Messages) and with ICMP Echo Replies.

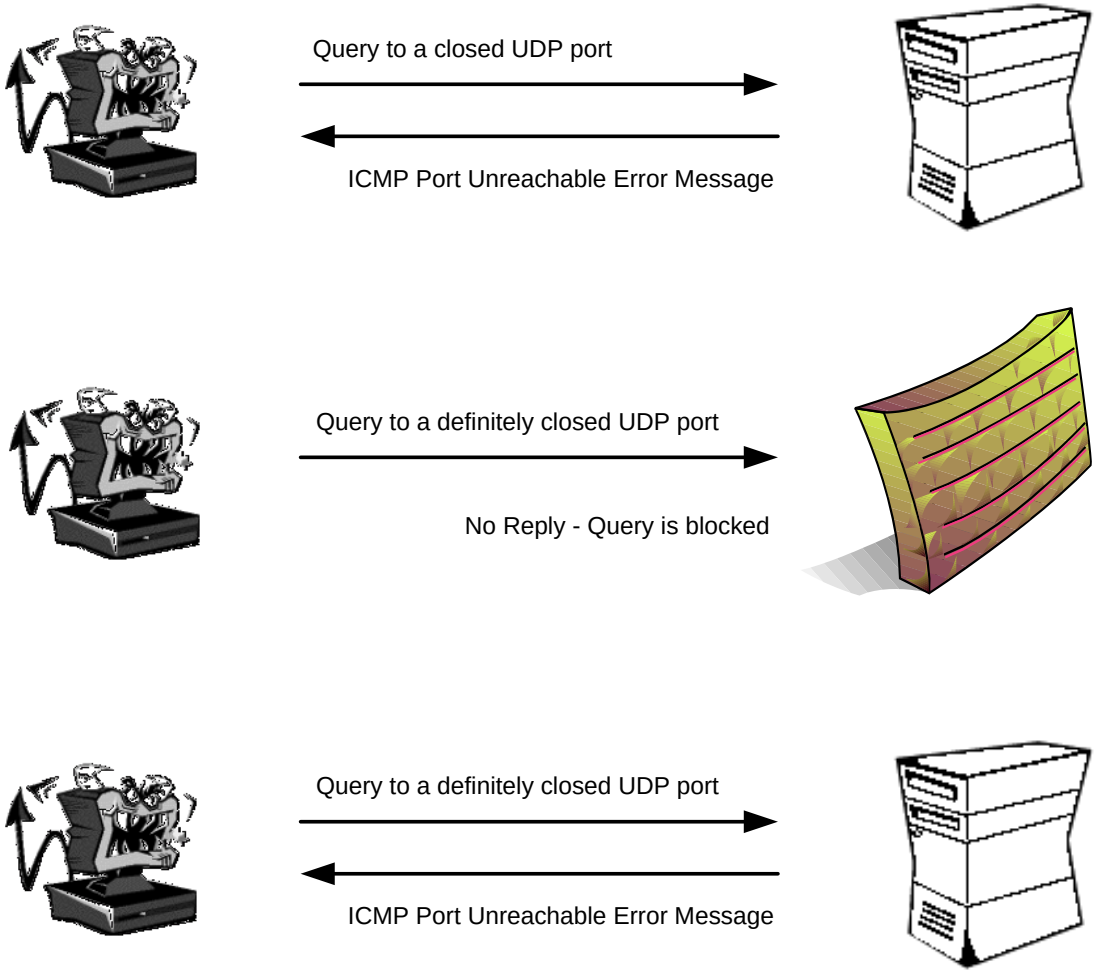
DF Bit Echoing

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?

IPID

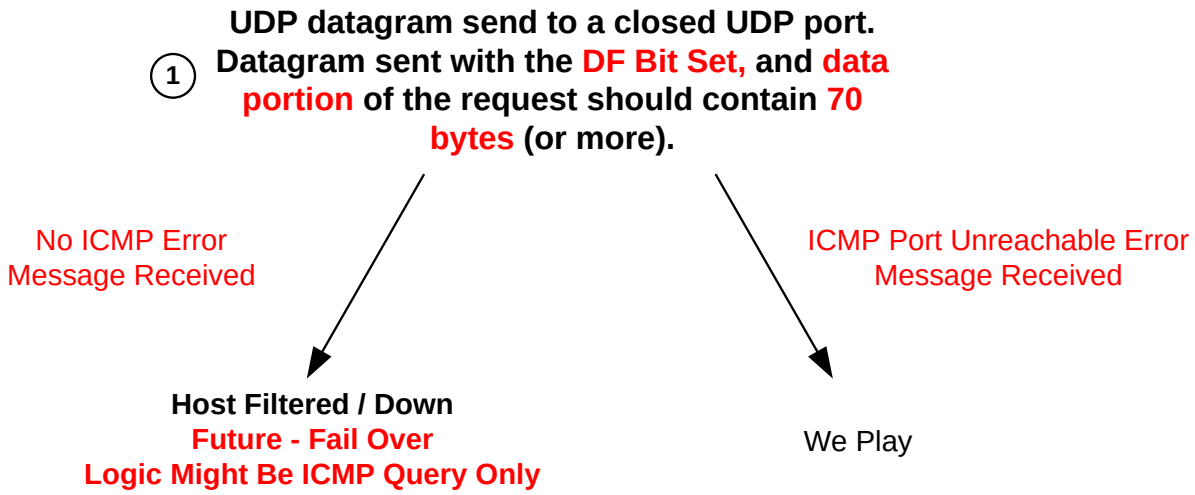
Linux Kernels 2.4.0 – 2.4.4 will send ICMP Echo replies (and requests) with an IP ID field value of 0.

Xprobe - How Do We Start?



Xprobe - How Do We Start?

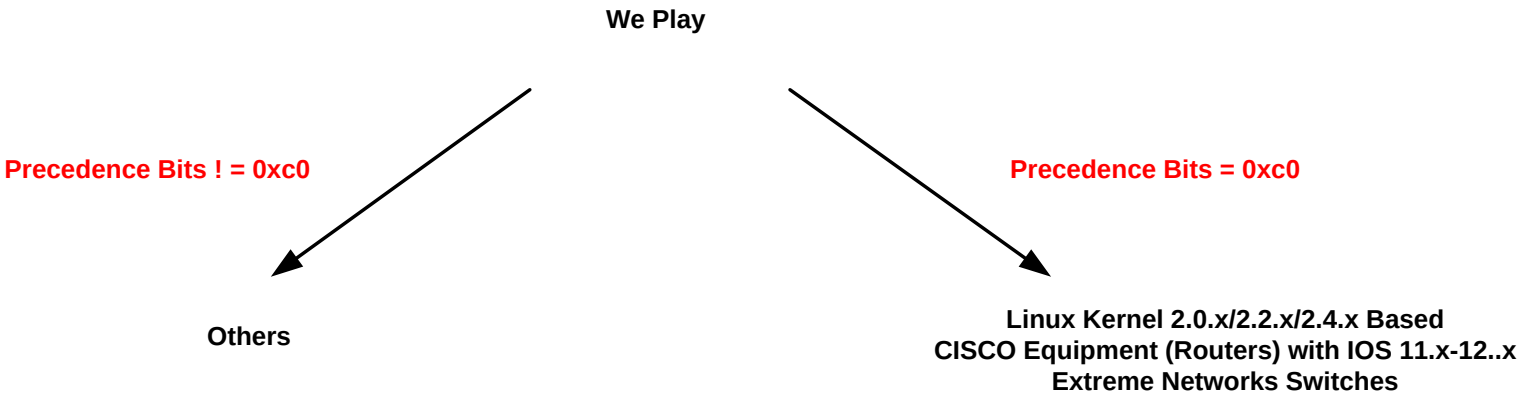
An example with the Static logic



- We query a definitely closed UDP port.
  http://www.isi.edu/in-notes/iana/assignments/port-numbers
- An indicator is being given for the presence of a Filtering Device
- If no ICMP Error Message is received, we might use the 'query only' logic
- The size of the Offending UDP datagram is 70 bytes

Xprobe - How Do We Start?

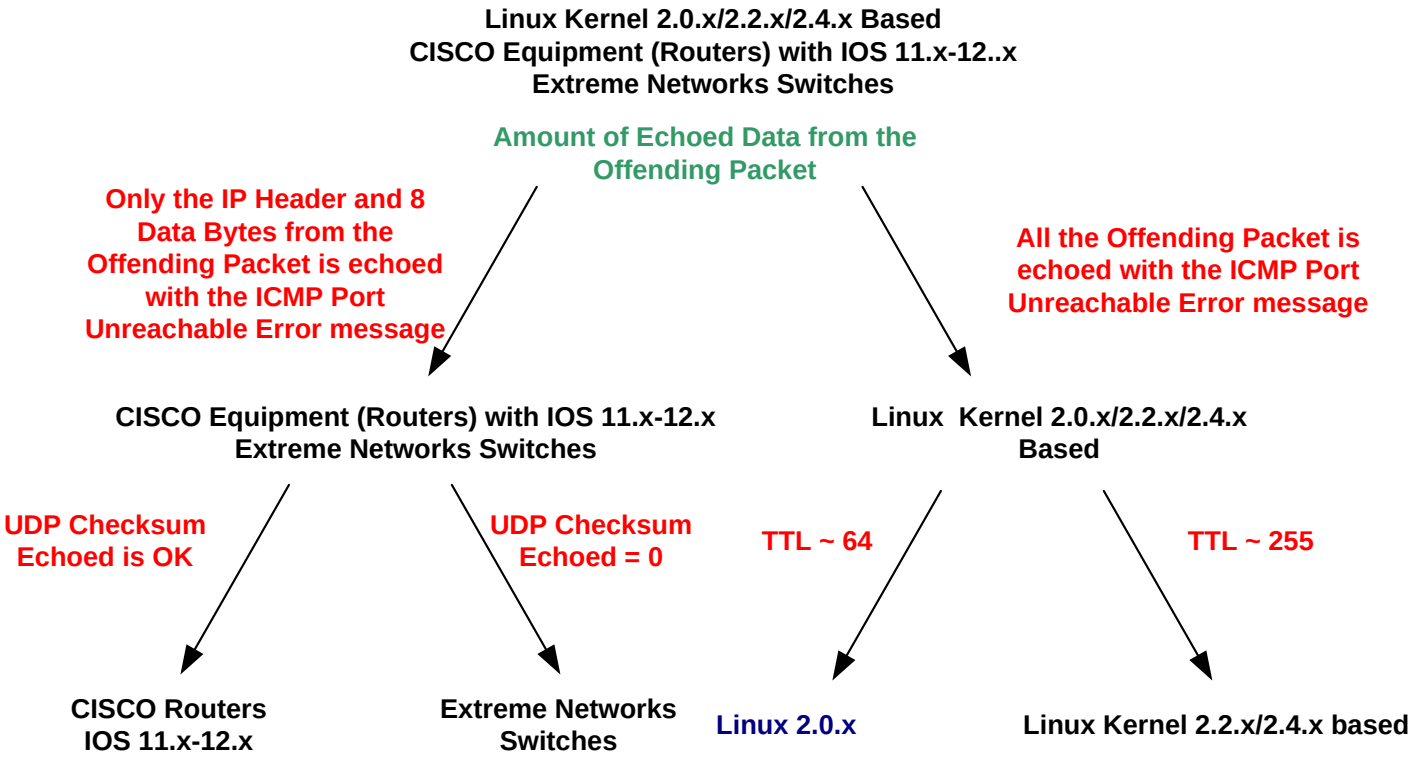
An example with the Static logic



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

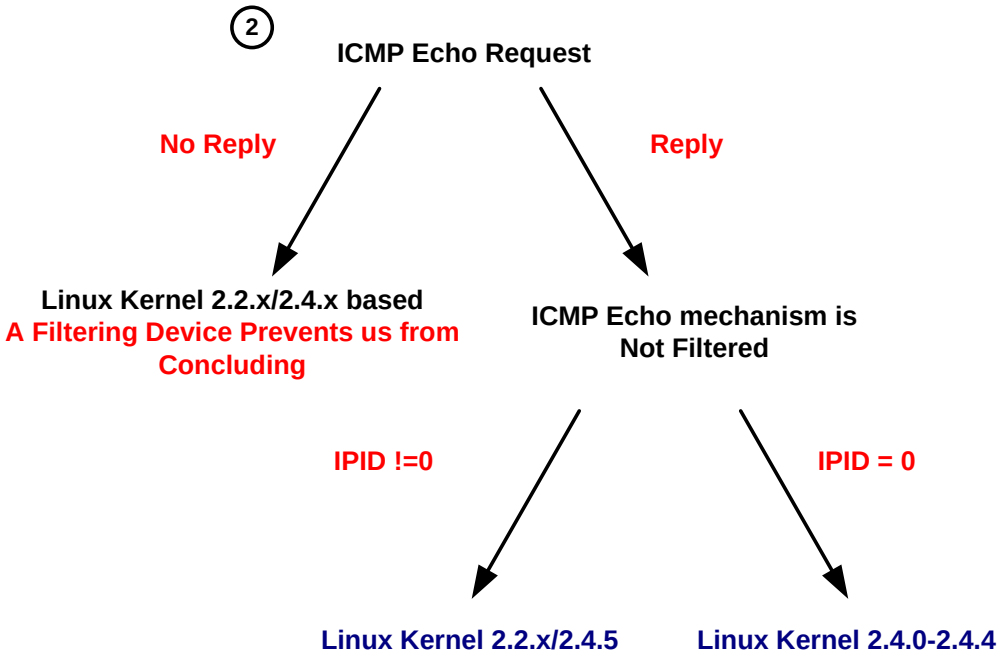
Xprobe - How Do We Start?

An example with the Static logic



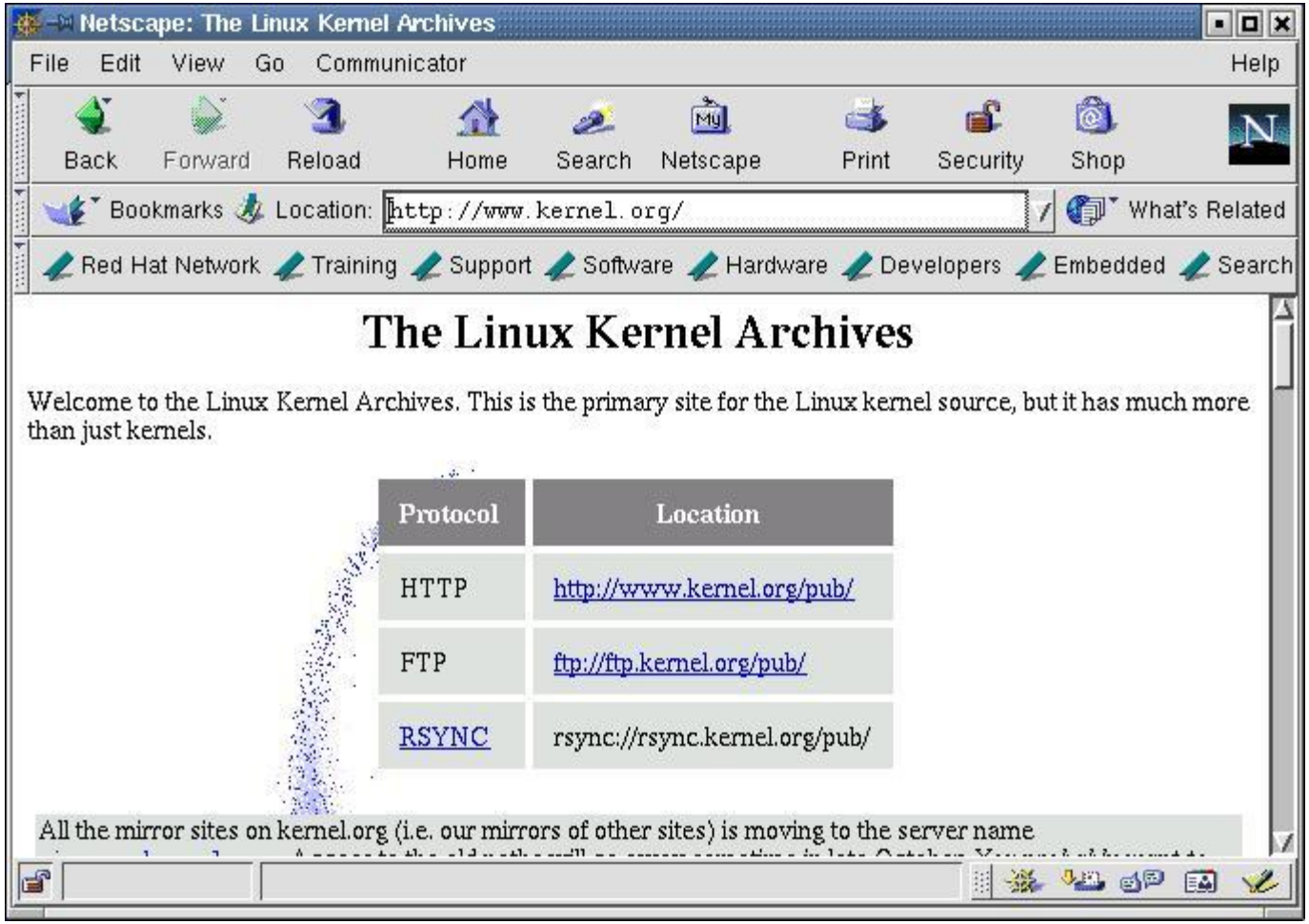
Xprobe - How Do We Start?

An example with the Static logic

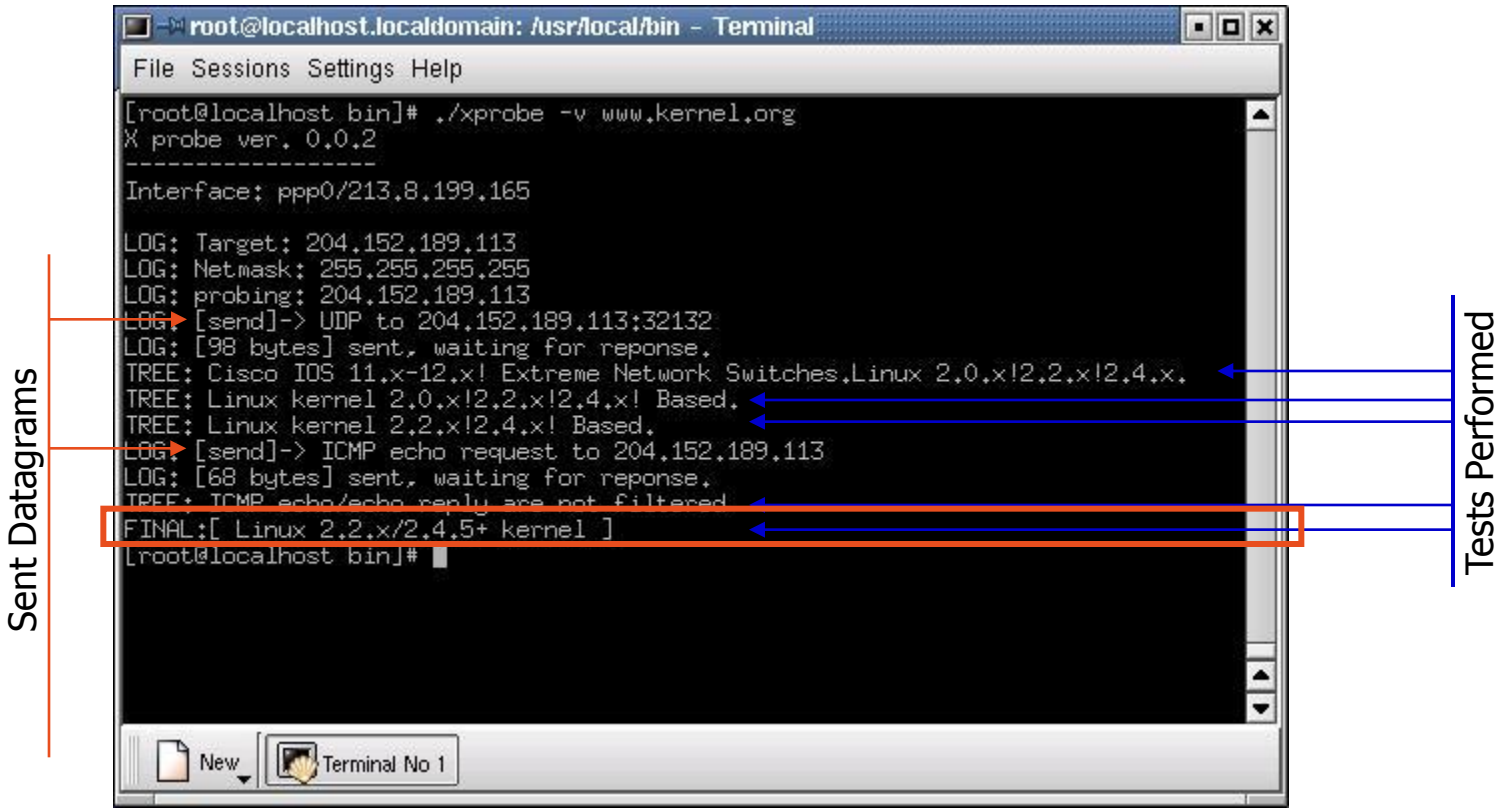


- Linux Kernel 2.4.0-2.4.4 will use 0 as its IPID field value with ICMP Query replies (this was later fixed with Linux Kernels 2.4.5 and above).
- Linux Kernel 1.x does not set the Precedence field value to 0xc0 with ICMP error messages.

Example – www.kernel.org



Example – www.kernel.org



Example – www.kernel.org

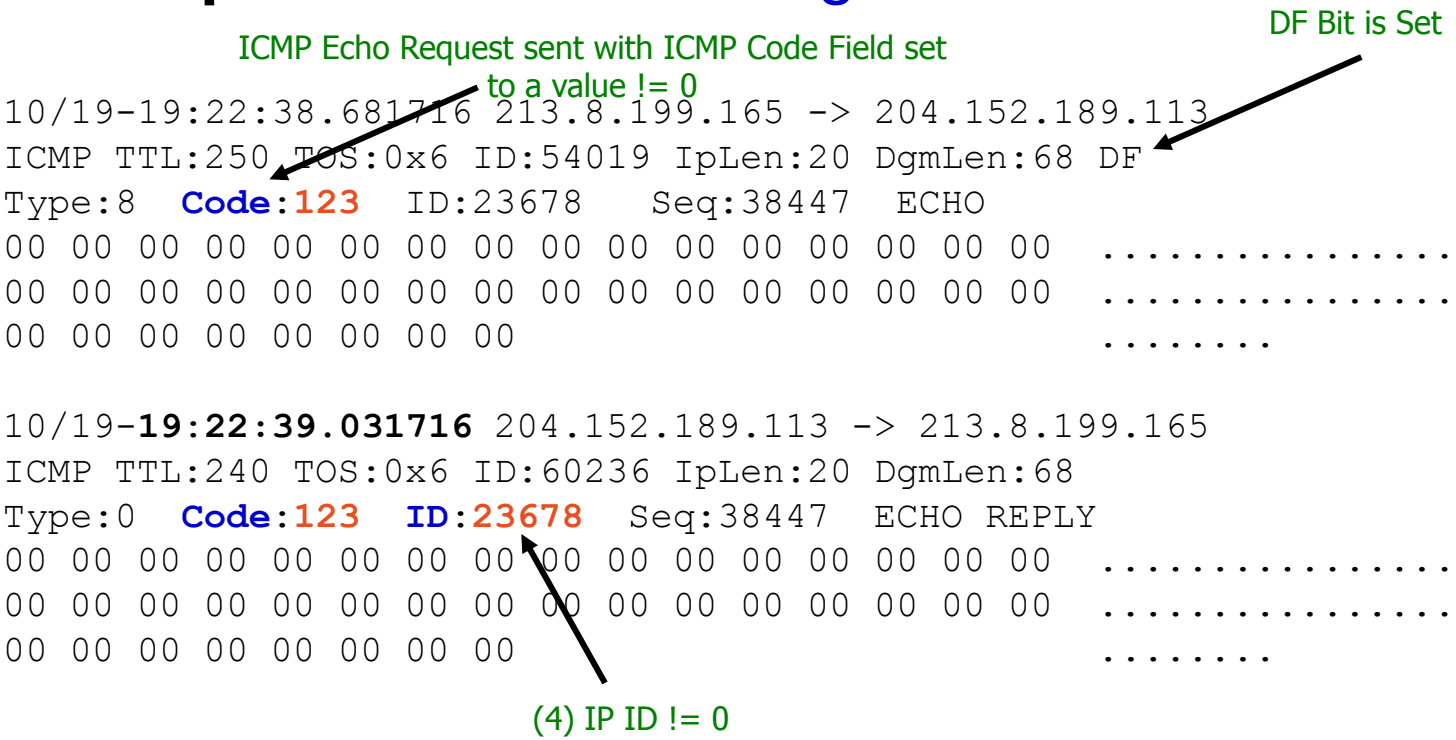
```
10/19-19:22:38.321716 213.8.199.165:14320 -> 204.152.189.113:32132
UDP TTL:250 TOS:0x0 ID:33305 IpLen:20 DgmLen:98 DF
Len: 78
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00                                ......

10/19-19:22:38.681716 204.152.189.113 -> 213.8.199.165
ICMP TTL:240 TOS:0xC0 ID:60235 IpLen:20 DgmLen:126
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
213.8.199.165:14320 -> 204.152.189.113:32132
UDP TTL:238 TOS:0x0 ID:33305 IpLen:20 DgmLen:98
Len: 78
** END OF DUMP
00 00 00 00 45 00 00 62 82 19 40 00 EE 11 E3 B8  ....E..b..@.....
D5 08 C7 A5 CC 98 BD 71 37 F0 7D 84 00 4E 23 25  .......q7.}..N#%
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00                                ......
```

(1) Precedence Bits Value = 0xC0

(2) Extra Data Echoed

(3) TTL

Example – www.kernel.org



Linux Based on Kernel 2.2.x/2.4.5+

Time Elapsed ~700ms

Xprobe - OS Identification List (Static)

- Microsoft Windows 95
- Microsoft Windows 98
- Microsoft Windows 98 SE
- Microsoft Windows ME
- Microsoft Windows NT4 SP3 and Below
- Microsoft Windows NT4 SP4 and UP
- Microsoft Windows 2000 (including SP1 and SP2)
- Microsoft Windows XP

Xprobe - OS Identification List (Static)

- Linux Kernel 2.0.x
- Linux Kernel 2.2.x (and 2.4.5+)
- Linux Kernel 2.4.0 – 2.4.4
- Sun Solaris 2.3 – 2.8
- Sun OS 4.x
- HPUX 10.x, 11.x
- MacOS 7.x-9.x
- AIX 3.x, 4.x
- Novell Netware

Xprobe - OS Identification List (Static)

- FreeBSD 2.x - 4.1, 4.1 - 4.3, 5.0 (future)
- BSDI 2.x, 3.x, 4.x
- NetBSD 1.x, 1.2.x, 1.3.x, 1.4.x, 1.5.x
- OpenBSD 2.1-2.3, 2.4-2.5, 2.6-2.9
- Ultrix
- OpenVMS
- DGUX / Compaq Tru64
- IBM OS/390

Xprobe - OS Identification List (Static)

- NFR Appliance
- Cabletron SSR 8000
- Cisco Routers with IOS 11.x-12.x
- Extreme Networks Switches

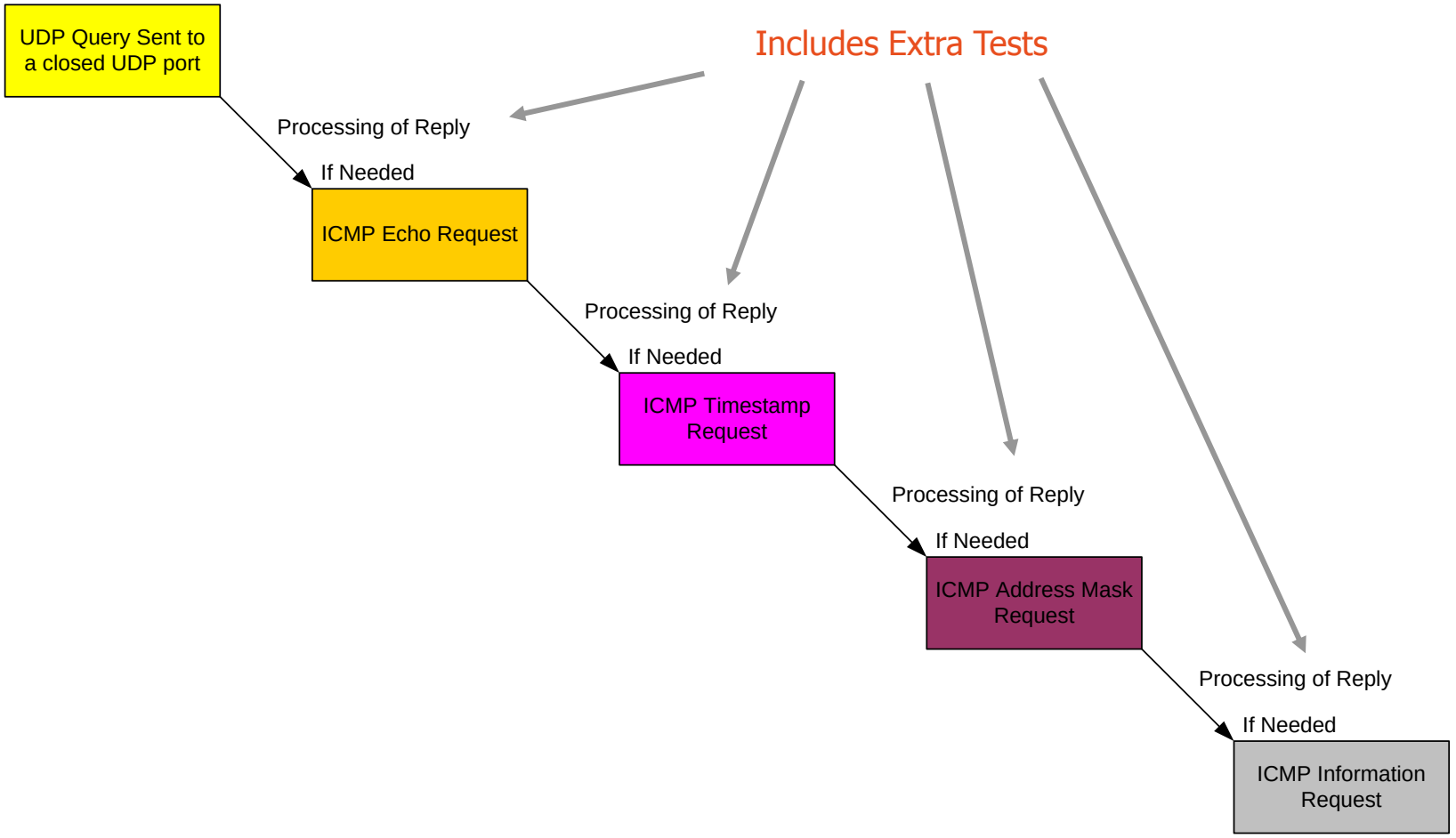
Xprobe - The Signature Based Approach

The Logic Of Initiation of Queries

- The initiation of queries with the static version of Xprobe (v0.0.x) is done according to the decision tree.
- Initiation of queries with the Signature based version of Xprobe (v0.x) currently has a certain logic.

Xprobe - The Signature Based Approach

The Logic Of Initiation of Queries



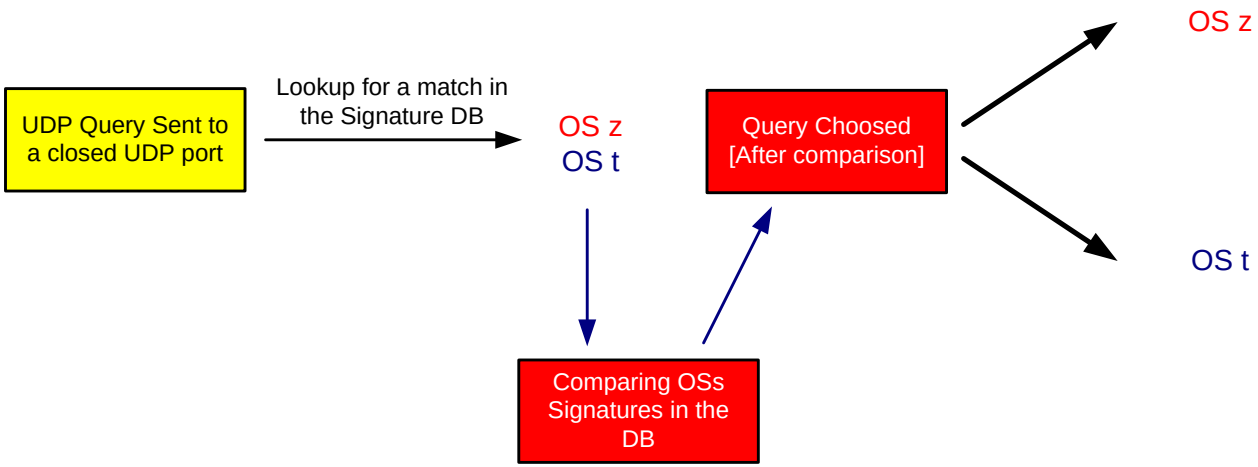
Xprobe - The Signature Based Approach

The Logic Of Initiation of Queries

In the future we will initiate queries according to specific differentiations.

This means that if we receive the exact response for our offending UDP query that matches two operating systems, for example, we will not automatically send an ICMP Timestamp request, but we will compare the two signatures in our database and look for the exact query that will give us the ability to differentiate between the two. This way we save bandwidth, and make the fingerprinting/manual detection harder.

With more than two matches for a response we will another algorithm/decision logic.



Xprobe - The Signature Based Approach

The Signature Base

```
platform: "Some OS v.1.2-1.3"
udptest:0xc0:8:BAD:<64:0:20+:FLP:0xc0:BAD:BAD
udptest1:0xd8:8:BAD:<64:0:20+:FLP:0xc0:BAD:BAD
icmpecho:ZERO:0xc0:<64:ZERO:FLP:0x40:BAD
icmpts: yes:<64:BAD:BAD
icmpaddr: no
icmpinforeq: no
```

The signature of an OS comprises of six tests, from whom five are different tests (udptest, udptest1, icmpecho, icmpts, icmpaddr, icmpinforeq).

Xprobe - The Signature Based Approach

udptest

```
udptest:<Precedence_Bits>:<Data_Bytes_Echoed>:<UDP_Checksum_Echoed>:<TTL>:<IP_ID>:<IP_Total_Length_Echoed>:<IP_Off_Bits>:<DF_Bit>:<IP_Header_Checksum_Echoed>:<IP_ID_Echoed>
```

```
udptest:<value-bitmask>:<value>:<ZERO|BAD|GOOD>:<(=|<|>)<value>:<ZERO|OK>:<value>(+|-):<FLP|OK>:<0|OK>:<ZERO|BAD|GOOD>:<ZERO|BAD|GOOD>
```

Xprobe - The Signature Based Approach

icmpecho

```
icmpecho:<ICMP_Code_In_ICMP_Reply>:<Precedence_Bits>:<TTL>:<IP_ID>:<IP_Off_Bits>:<DF_Bit>
```

```
Udptest:<ZERO|OK>:<bitmask>:<(<|=|>)<value>:<ZERO|OK:0|OK>:<bitmask>:<NO|YES>
```

Xprobe - The Signature Based Approach

icmpts, icmpaddrreq, icmpinfo

```
icmpaddr:<Answer?>:<TTL>:<IP_ID>:<DF_Bit>
```

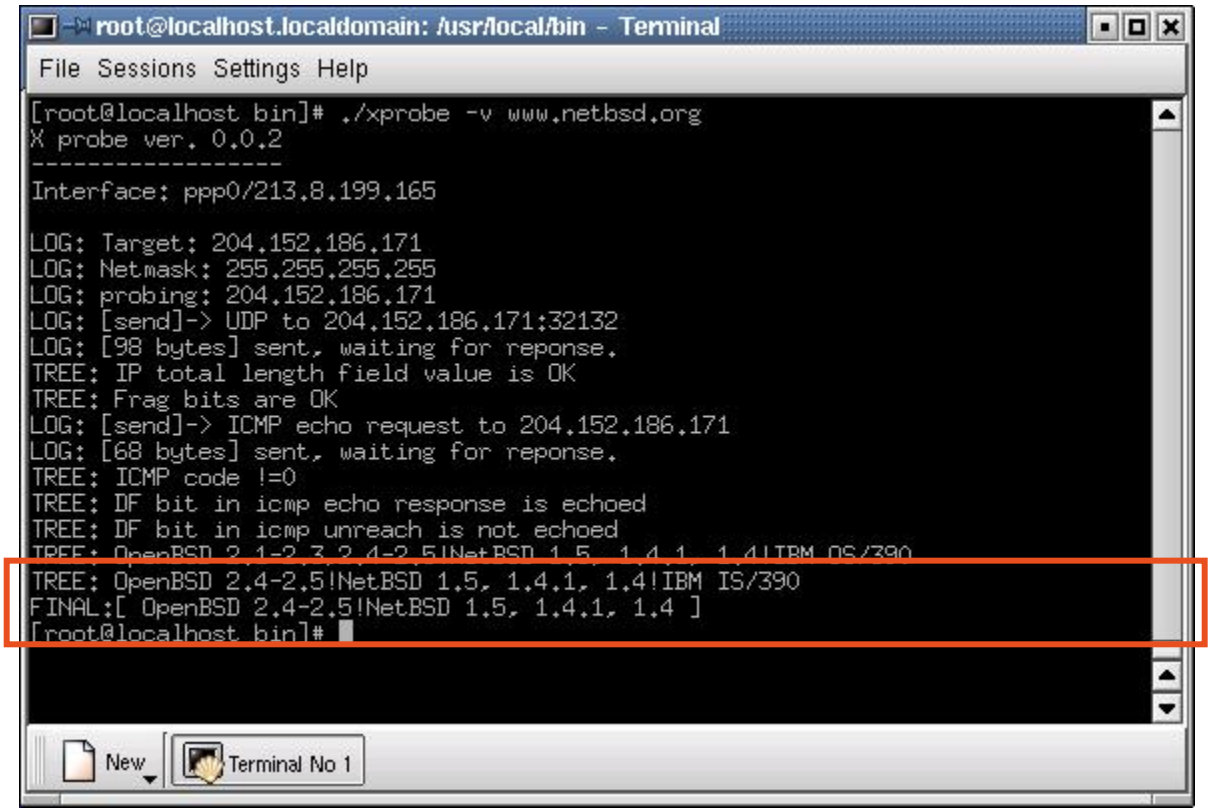
```
icmpaddr:<YES|NO>:<(<|=|>)<value>:<ZERO|OK>:<No|YES>
```

Xprobe - More Examples

www.netbsd.org

Xprobe - More Examples

www.netbsd.org

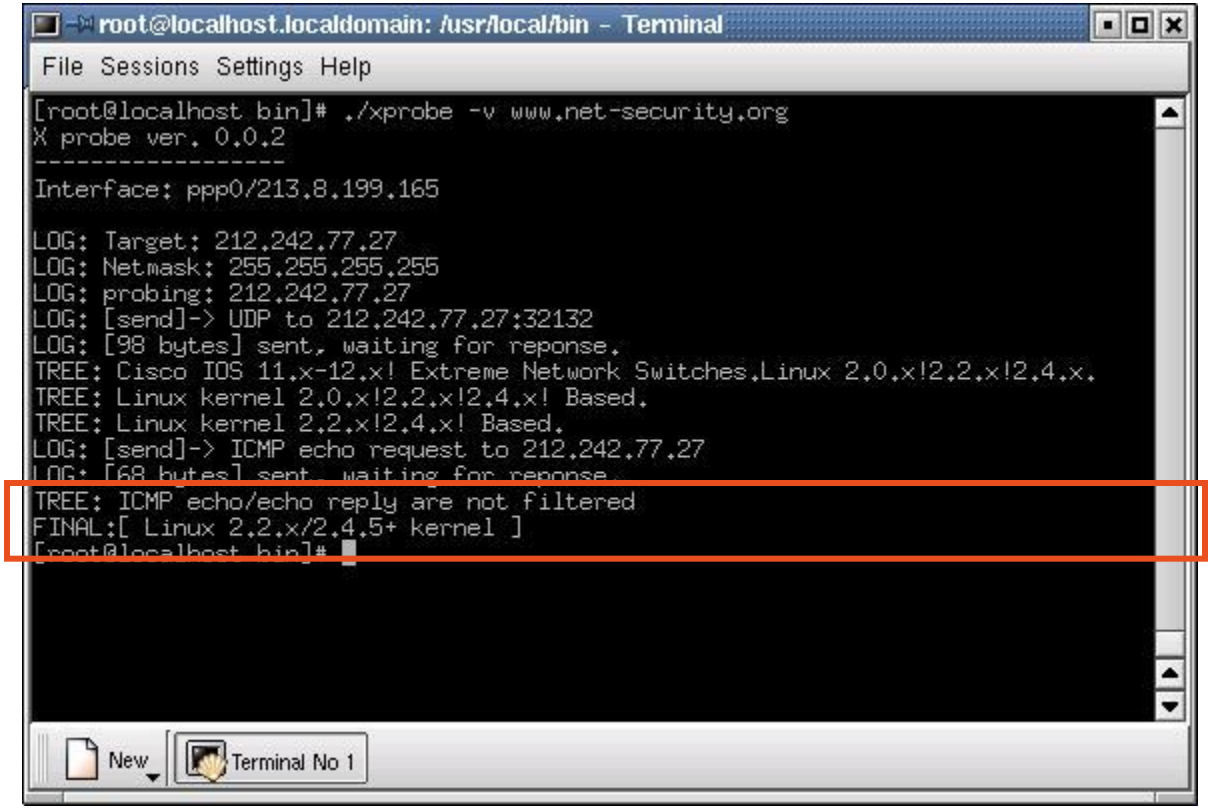


Xprobe - More Examples

www.net-security.org

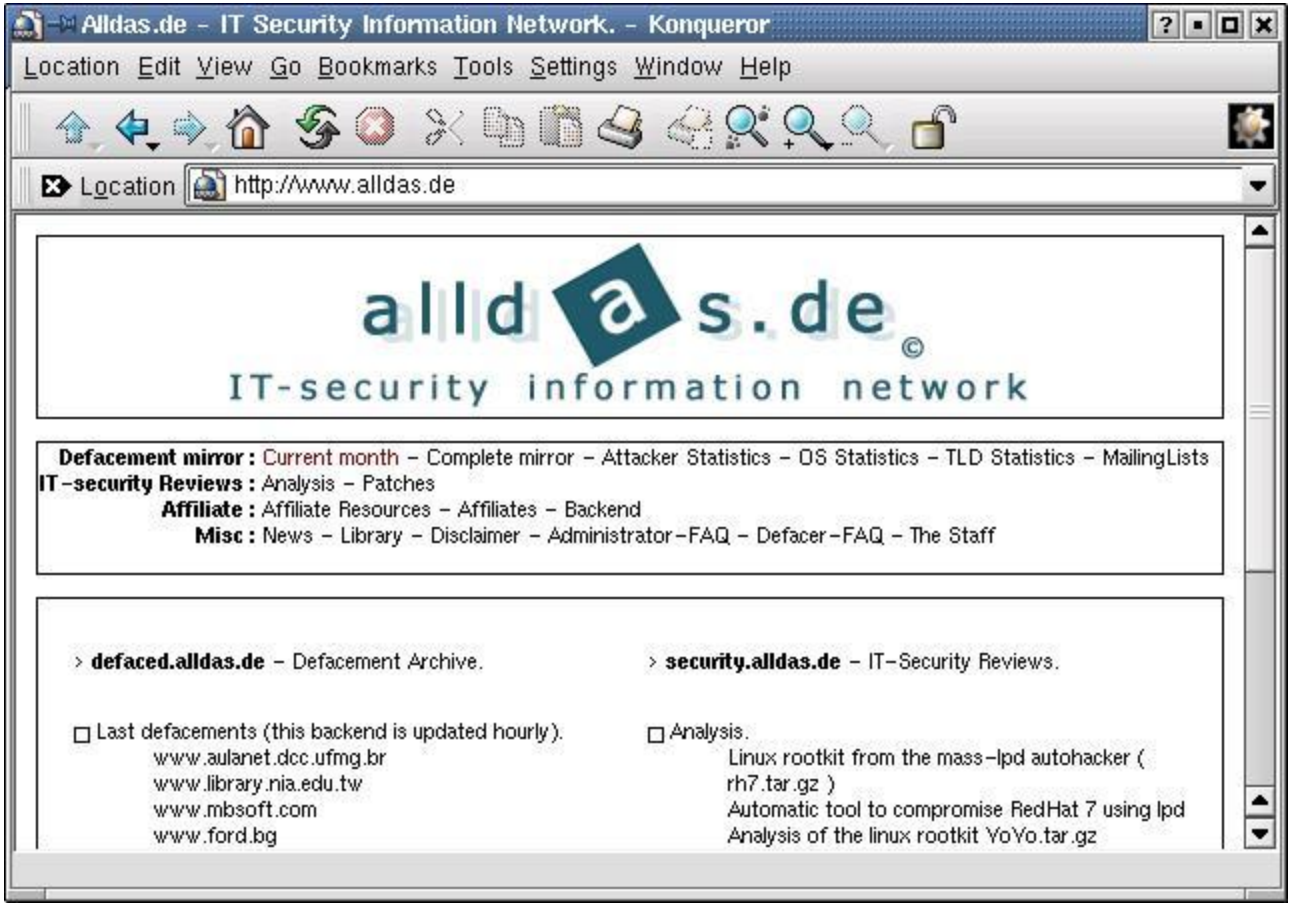
Xprobe - More Examples

www.net-security.org



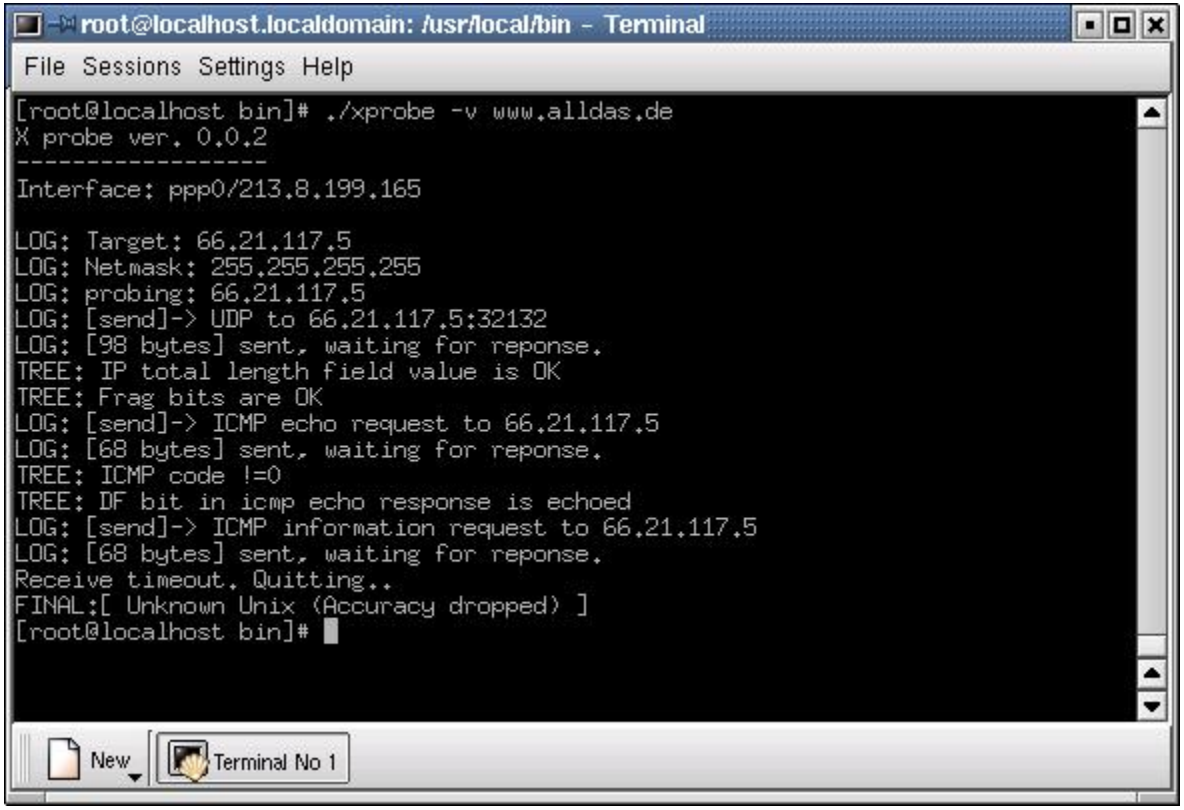
Xprobe - More Examples

www.alldas.de



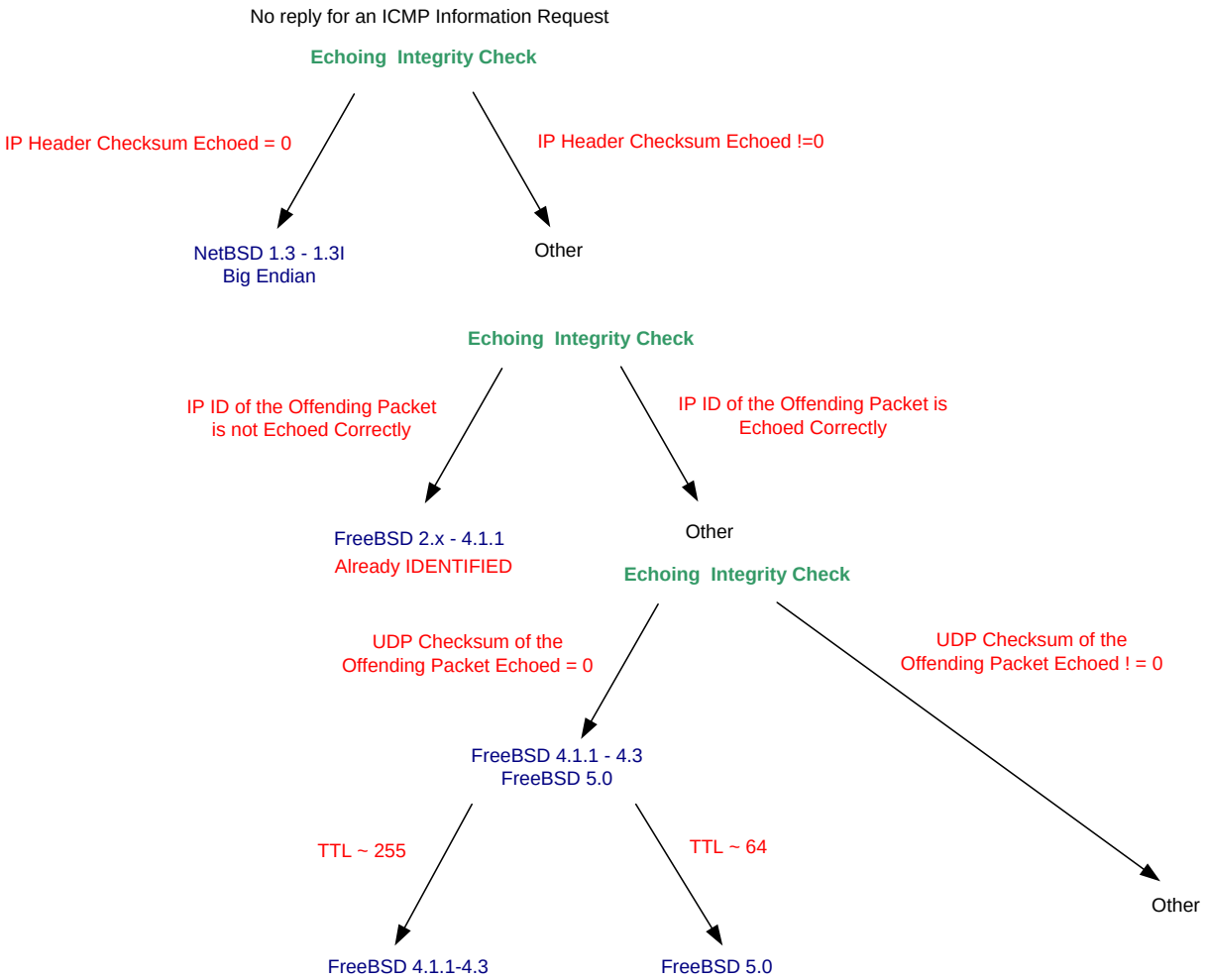
Xprobe - More Examples

www.alldas.de



Xprobe - More Examples

www.alldas.de



Xprobe - More Examples

www.alldas.de

IPID Echoed OK

```
10/19-19:14:00.871716 213.8.199.165:24493 -> 66.21.117.5:32132
UDP TTL:250 TOS:0x0 ID:56241 IpLen:20 DgmLen:98 DF
Len: 78
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00                                ......
```

TTL < 64

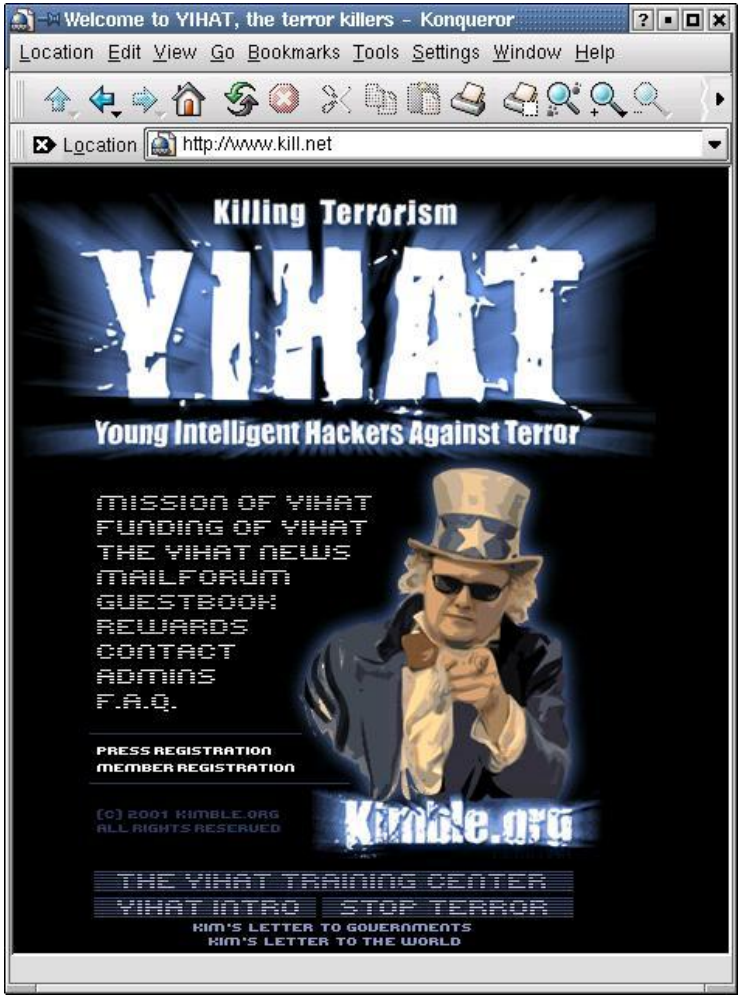
```
10/19-19:14:01.191716 66.21.117.5 -> 213.8.199.165
ICMP TTL:41 TOS:0x0 ID:49572 IpLen:20 DgmLen:56 DF
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
213.8.199.165:24493 -> 66.21.117.5:32132
UDP TTL:233 TOS:0x0 ID:56241 IpLen:20 DgmLen:98
Len: 78
** END OF DUMP
00 00 00 00 45 00 00 62 DB B1 40 00 E9 11 62 10  ....E..b..@...b.
D5 08 C7 A5 42 15 75 05 5F AD 7D 84 00 4E 00 00  ....B.u._.}..N..
```

UDP Checksum 0

FreeBSD 5.0 (automatically supported by the 0.x version)

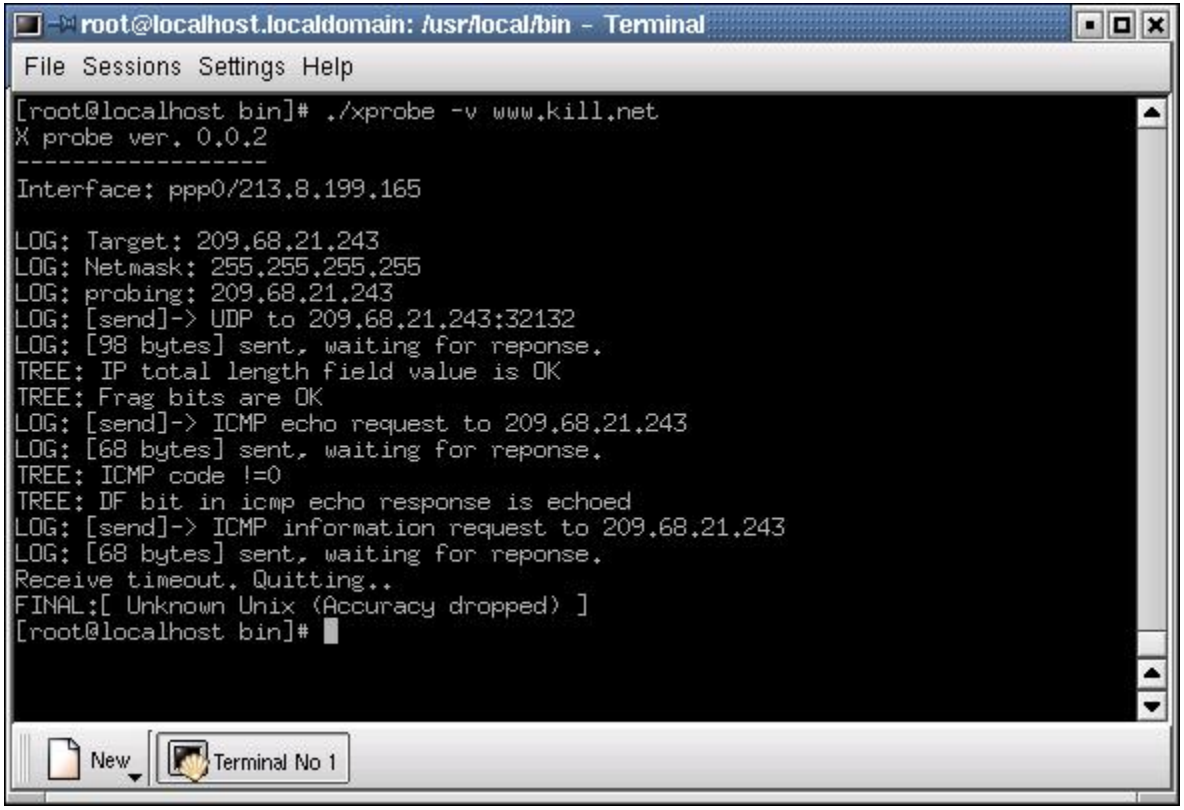
Xprobe - More Examples

www.kill.net



Xprobe - More Examples

www.kill.net



Xprobe - More Examples

www.kill.net

IPID Echoed OK

```
10/19-18:59:00.911716 213.8.199.165:6314 -> 209.68.21.243:32132
UDP TTL:250 TOS:0x0 ID:54470 IpLen:20 DgmLen:98 DF
Len: 78
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00                                ......
```

TTL < 255

```
10/19-18:59:01.211716 209.68.21.243 -> 213.8.199.165
ICMP TTL:238 TOS:0x0 ID:40233 IpLen:20 DgmLen:56 DF
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
213.8.199.165:6314 -> 209.68.21.243:32132
UDP TTL:238 TOS:0x0 ID:54470 IpLen:20 DgmLen:98
Len: 78
** END OF DUMP
00 00 00 00 45 00 00 62 D4 C6 40 00 EE 11 33 DE  ....E..b..@...3.
D5 08 C7 A5 D1 44 15 F3 18 AA 7D 84 00 4E 00 00  .....D....}..N..
```

UDP Checksum 0

FreeBSD 4.1.1 – 4.3 (automatically supported by the 0.x version)

Xprobe - Known Problems

- Signature Base Needs to Grow
- No ids evasion is done yet. packets are easy to fingerprint. once core features developed, optional 'masking' of payload data will be done. (ICMP echo request like the once produced with the ‘ping’ utility, DNS queries etc).
- ICMP Echo request is sent with a code field != 0 (still nobody looks at this parameter).

Further Reading

ICMP Usage In Scanning, v3.0 by Ofir Arkin,
http://www.sys-security.com

X – Remote ICMP based OS Fingerprinting Techniques, by Fyodor Yarochkin and Ofir Arkin,
http://www.sys-security.com

RFC 792: Internet Control Message Protocol,
http://www.ietf.org/rfc/rfc0792.txt

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

RFC 1256: ICMP Router Discovery Messages,
http://www.ietf.org/rfc/rfc1256.txt

RFC 1349: Type of Service in the Internet Protocol Suite,
http://www.ietf.org/rfc/rfc1349.txt

RFC 1812: Requirements for IP Version 4 Routers,
http://www.ietf.org/rfc/rfc1812.txt

Tools Used

Xprobe written by Fyodor Yarochkin & Ofir Arkin
http://www.sys-security.com
http://www.notlsd.net/xprobe
http://xprobe.sourceforge.net

tcpdump
http://www.tcpdump.org

Snort written by Marty Roesch
http://www.snort.org