Techniques in OS-Fingerprinting

an exposition by

NOSTROMO

Hagenberg, September 2005

Contents

Abstract

This exposition will give a little insight into the working principles of TCP operating system (OS) fingerprinting applications. In chapter 2 the basics of TCP and ICMP fingerprinting mechanisms, as well as more simple ways to get the operating system an aimed device is running on, are described. These include direct and indirect banner grabbing. Also specialised applications such as `nmap` and `Xprobe2` are described in this chapter. Chapter 3 will point out some ways to defeat fingerprinting probes on `UNIX`-based and `Linux` operating systems.

Preface

Acknowledgements and Trademarks

`UNIX` is a registered trademark, licensed exclusively through The Open Group.
`Linux` is a registered trademark of Linus Torvalds.
`Windows` is a registered trademark of Microsoft Corporation.
`Nmap` is a registered trademark of Insecure.Com LLC.
Other terms, which are trademarks, are the property of their respective owners.

In this document the term "UNIX" is used as a generic description for UNIX- like operating systems of different manufacturers.

Chapter 1

Introduction

1.1 What is OS fingerprinting?

OS fingerprinting describes the method of utilising gathered information of a target host to find out what operating system the machine is running on.

Or, as wikipedia[1] would describe it:

> TCP/IP stack fingerprinting (or OS fingerprinting) is the process in computing of determining the identity of a remote host's operating system by analyzing packets from that host.

1.2 Why OS fingerprinting?

When doing penetration testing today the tester starts to gather as much information of the target machine as possible. One major key information is the operating system the target is running on. As long as this information is not revealed, the attacker is limited in the variety of attacks, probes and exploits. Therefore the focus on initial information gathering is put on finding out the operating system.

There are several approaches to finding out the running operating system of an unknown host without having an account or any other way of logging in directly on this machine. Their range is from simple banner grabbing to highly sophisticated TCP- and/or ICMP-header analyses. This exposition will give a rough overview on some of them with a little in-depth dissection and description.

[1] http://en.wikipedia.org/wiki/OS_fingerprinting

Chapter 2

Techniques of OS fingerprinting

2.1 Non-automated techniques

2.1.1 Direct Banner Grabbing

Though banner grabbing is the most basic and easiest form of OS fingerprinting, it is often quite efficient and reliable. The approach is simple and doesn't require any special tools most of the time (cf. [Fyo98]).

Here is an example using `telnet`, a standard tool that can be found on all Microsoft `Windows` and `UNIX`-like platforms:

```
root@nostromo# telnet mail.fh-hagenberg.at 143
Trying 193.170.124.96...
Connected to postman.fh-hagenberg.at.
Escape character is '^]'.
* OK Microsoft Exchange Server 2003 IMAP4rev1 server version ↲
  6.5.7226.0 (postman.fhs-hagenberg.ac.at) ready.
```

When analyising the output in bits and pieces a lot of information is revealed from the single line that was returned by the server.

- *Microsoft Exchange Server 2003*: It is a Microsoft *Exchange Server 2003*.
- *IMAP4rev1 server*: It runs IMAP4.
- *version 6.5.7226.0*: It is version 6.5.7226.0.

Now it's up to an attacker to find an exploit for this specific version of the `Microsoft` *Exchange Server 2003*.

Another example will be given:

```
root@nostromo# telnet nostromo.joeh.org 80
Trying 193.170.32.26...
[...]
HEAD / HTTP/1.0
[...]
Server: Microsoft-IIS/8.1
[...]
```

The analysis would reveal:

- *Server: Microsoft-IIS/8.1*: It is a `Microsoft` *IIS*, version 8.1.

A search on *Google*[1] would show that there is no *IIS* of version 8.1. So here it is obvious that the banner has been faked in some way.
Faking is often linked to the phrase "Security by Obscurity". Throughout this exposition it will be shown that banner string faking is not a way of keeping someone who is familiar with OS fingerprinting techniques, to get a reliable result. OS fingerprinting in the majority of cases is always a combination of the output from as many probes/tools as possible, so faking just one banner output will not be enough.

2.1.2 Indirect Banner Grabbing

Banners are often revealed in an indirect way too. For example email headers often contain the version string of the client that is used by a user – sometimes including the operating system version. Sometimes even the firewall that an email passed through adds its banner to the header of the email. Same with servers that forwarded the message, and the local delivery agent (if present). Information that is often revealed by headers is an involved virus scanner too.
Another example would be the `SYST` command in FTP. Doing a quick test on ftp://ftp.microsoft.com and ftp://ftp.debian.org returned "`215 Windows_NT`" and "`215 UNIX Type: L8`". Though these banners do not reveal the exact underlying operating system it can give a first hint on the OS family that is used (like `Microsoft Windows`, `Linux`, a `UNIX`-derivat, `Apple Macintosh`, etc.) [Fyo98].
One more way to get information is analyzing offered files, like trying to download `/bin/ls` or `/bin/gunzip` from a ftp server:

[1]http://www.google.com

```
root@nostromo# wget ftp://ftp3.ru.freebsd.org/bin/gunzip
[...]
18:23:19 (104.25 KB/s) - `gunzip' saved [46836]
root@nostromo# file gunzip
gunzip: ELF 32-bit LSB executable, Intel 80386, version 1    ↲
  (FreeBSD), for FreeBSD 5.3, dynamically linked (uses       ↲
  shared libs)
```

2.2 TCP and ICMP fingerprinting

2.2.1 The TCP packet header

In order to understand the technique of TCP fingerprinting (which `nmap` heavily relies on – see Chapter 2.3.1, page 6) a little insight into a TCP header is necessary. Therefore a TCP header will be shown here:

4	8	12	16	20	24	28	32
Sequence Number							
Acknowledgement Number							
Offset	Reserved		Flags	Window Size			
Checksum				Urgent Pointer			
Options						Padding	
Data							

Figure 2.1: TCP header fields [Hun92]

Especially the flags header field is of special interest for the fingerprinting application. The six flags in detail are:

"`URG`" (1 bit):	Urgent pointer field significant.
"`ACK`" (1 bit):	Acknowledgment field significant.
"`PSH`" (1 bit):	Push Function.
"`RST`" (1 bit):	Reset the connection.
"`SYN`" (1 bit):	Synchronize sequence numbers.
"`FIN`" (1 bit):	No more data from sender.

[JBB92]

These flags are essential for OS fingerprinting since each operating system reacts differently to normal and special crafted TCP packets sent to its network stack. Especially tools as described in chapter 2.3 make heavy use of those flags.

2.2.2 The ICMP packet header

In order to understand the working principles of ICMP fingerprinting and `Xprobe2` (described in chapter 2.3.2, page 8) a little insight into an ICMP header is necessary. Therefore a prototype ICMP header will be shown here:

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Data							

Figure 2.2: ICMP header fields [Hun92]

The `Type` field specifies the format of the ICMP message. Most common types are:
- 3: Destination Unreachable,
- 4: Source Quench,
- 5: Redirect Message,
- 8 and 0: Echo Request and Echo Reply,
- 11: Time Exceeded,
- 12: Parameter Problem,
- 13 and 14: Timestamp Request and Timestamp Reply,
- 15 and 16: Information Request and Information Reply.

The header looks different for each request and reply packet though. The type field specific headers will be shortly described in appendix **??**, page **??** and shortly described in chapter 2.3.2, page 8.

2.3 Automated techniques

Of course all these methods described in chapter 2.1.1 can be automized. Two applications that have gained fame on the field of OS fingerprinting are *nmap* and *Xprobe2*. This exposition will take a closer look at *nmap* and will give a little insight into the way fingerprinting is done. Furthermore a quick overview of how *Xprobe2* does OS detection will be given too.

2.3.1 *nmap*

nmap was written by Fyodor[2]. It can be obtained from http://www.insecure.org/nmap/ and it was published under the terms of the GNU General Public License[3] as published by the *Free Software Foundation.*

nmap begins it's OS detection by sending an ICMP ping request to the target, then it connects to port 80 (HTTP) to see if the target is responding and running at all[4]. Then *nmap* does the actual portscan, searching for at least one open[5] and one closed[6] port. To gain exact information about the underlying OS *nmap* sends several special crafted TCP packets – at least four packets are sent to an opened port, and three to a closed port – and records the replies. It then makes a lookup in the OS-detection fingerprint file.

The first sent packet contains only the SYN flag set. This is typically done when initiating a TCP connection. The second packet has no flag set at all, which is usually refered to as a null scan, whilst for the third scan the URG, PSH, SYN and FIN flag are set. This combination of flags is not illegal though very unusual (RFC-1025 calls this sort of crafted packet a "kamikaze", "nastygram", "christmas tree" or "lamp test" packet [Pos87]). Furthermore several options are set in the TCP packet header (see Table 2.1, page 4) that again are interpreted differently by operating systems.

So all these probes are evaluated and looked up in the nmaps OS-fingerprints file. This file ("`nmap-os-fingerprints`") contains a pattern that can easily be extended for new operating systems since they all are based on one template. Here is an example for "FreeBSD 5.3-RELEASE #0":

```
# FreeBSD 5.2-CURRENT (Jan 2004) on x86
# FreeBSD 5.2.1-RELEASE i386
# FreeBSD 5.3-RELEASE #0
Fingerprint FreeBSD 5.2 - 5.3
Class FreeBSD | FreeBSD | 5.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=100HZ)
T1(DF=Y%W=FFFF%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=FFFF%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
```

[2]"Fyodor" actually is the unknown hackers favourite author called Fyodor Dostoyevsky. He (or she) decided to stay anonymous and therefore uses this handle [Fyo05].

[3]http://www.gnu.org/

[4]Today it is common practice to drop ICMP requests, so it looks like the host is down. As shown in chapter 2.3.2, page 8 this makes sense and can improve security

[5]"open" means an application is listening and awaiting connections.

[6]"closed" means no application is listening on this specific port.

```
12 T6(DF=Y%W=0%ACK=0%Flags=R%Ops=)
13 T7(DF=Y%W=0%ACK=S%Flags=AR%Ops=)
14 PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=0%         ↲
15   ULEN=134%DAT=E)
```

- Line 1 to 3: are comments, describing what is covered by this fingerprint.
- Line 4: specifies again what this fingerprint covers.
- Line 5: gives another rough overview of what is scanned (in this case it is *FreeBSD*, but it could as well be a *Zyxel*, *Cisco*, *Netgear*, etc. device).
- Line 6: gives the predictability of the used TCP initial sequence number (ISN[7]). Further details provided are: `Class=TR` describes the ISN as truly random (”`TR`”). `gcd=<6` means the greatest common denominator (”`gcd`”) is less than 6. ”`%`” divides sub-tests (”`&`” would be a logical AND, ”`|`” a logical OR operation). `IPID=I` describes the IPID[8] to be increased by a standard increment (”`I`”) with each sent packet. `TS=100HZ` means the timestamp (”`TS`”) increases by 100 every second.
- Line 7: specifies test 1 (”`T1`”) more in detail. `DF=Y` specifies the status of the Don't Fragment (”`DF`”) flag (it is either ”N” (No), or ”Y” (Yes)). `W` specifies the window size (”`W`”) received from the reply. `ACK=S++` refers to the expected ACK value, which was the initial sequence's ACK number plus one (”`S++`”), other possible return values could be `S` which indicates that the number returned was the sequence number sent, or `O` which indicates some other value was returned.. `Flags=AS` displays the TCP flags that are enabled in the reply, namely the Acknowledgment (”`A`”) and the Synchronize (”`S`”) flag, other values could be `R` which means the Reset flag was set. `Ops=MNWNNT` displays the TCP options that are enabled in the reply, namely MNWNNT (”`M`” is Maximum Segment Size, ”`N`” is No Op, ”`W`” is Window Scale, ”`T`” is Timestamp.
- Line 8: specifies test 2 (”`T2`”) more in detail. `Resp=N` means no response to the specified probe was received from the target host.
- Line 9 to 13: specifies test 9 (”`T2`”) through test 13 (”`T13`”) more in detail. For the rest of the options, see previous lines.
- Line 14/15: specifies the Port Unreachable test (”`PU`”) more in detail. For most of the values see specifications for Line 6 to 13. `TOS=0` displays the type of service (”`TOS`”) as a hexadecimal value. `IPLEN=38`

[7]The principle here is to find patterns in the initial sequence numbers chosen by TCP implementations when responding to a connection request.

[8]The IPID attribute refers to the IP identification bytes in the IP header.

displays the IP datagram total Length ("IPLEN") in bytes from the response packet's IP header as a hexadecimal value. RIPTL=148 shows the Repeated back IP datagram Total Length (RIPTL), which is the IP datagram total length from the echoed IP header – this value is referenced from the IPLEN attribute. RID=E displays the IP identification bytes in the reply (RID) that are identical to the original frame ("E"). RIPCK=E means a comparison of the Returned IP Checksum (RIPCK) to the checksum of the sent packet did equal ("E"). UCK=0 means that the UDP Checksum (UCK) of the returned packet did equal ("E") to the one in the sent probe. ULEN=134 means that the UDP Length (ULEN) in the response frame matches the sent value ("134") from the original frame. DAT=E displays if the Data (DAT) in the returned packet did equal ("E") to the original sent frame.

Here is the example according to above in practice:

```
root@nostromo# nmap -F -sT -O 193.170.32.26
[...]
Interesting ports on nostromo.joeh.org (193.170.32.26):
[...]
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
113/tcp  open  auth
443/tcp  open  https
706/tcp  open  silc
Device type: general purpose
Running: FreeBSD 5.X|6.X
OS details: FreeBSD 5.2 - 5.4, FreeBSD 5.2-CURRENT - 5.3      ↲
  (x86) with pf scrub all, FreeBSD 5.2.1-RELEASE or           ↲
  6.0-CURRENT
[...]
```

2.3.2 *Xprobe2*

Xprobe2 was written by Fyodor Yarochkin[9], Ofir Arkin and Meder Kydyraliev. It can be optained from http://xprobe.sourceforge.net/ and it was published under the terms of the GNU General Public License as published by the *Free Software Foundation.*
Unlike *nmap*, *Xprobe2* does not work with TCP packets, but with ICMP packets to collect the data needed for OS fingerprinting. It makes use of modules so it is extendable by developers and users at any time and can

[9]This Fyodor is not the same as the Fyodor from nmap (see 2.3.1, and http://www.snort.org/docs/faq/1Q05/node3.html

easily adopt new probes. After compiling the source code, provided at http://xprobe.sourceforge.net/, *Xprobe2* comes with only three files: the binary itself, a configuration file and a manual page (`man`). Besides the two "reachabilty" modules provided, only the first five of nine modules, named A to G plus two more modules that deal with TCP and UDP, for OS detection are of major interest here now. Their input is described in the configuration file ("`xprobe2.conf`") and a closer look at the section for "FreeBSD 5.3-RELEASE #0" from this file will be taken here:

```
1  [...]
2  OS\ID = "FreeBSD 5.3"
3  [...]
4  #Module A [ICMP ECHO Probe]
5  icmp_echo_reply = y
6  icmp_echo_code = !0
7  icmp_echo_ip_id = !0
8  icmp_echo_tos_bits = !0
9  icmp_echo_df_bit = 1
10 icmp_echo_reply_ttl = <64
```

- Line 2 names the operating system that will be described.
- Line 4 describes the ICMP type used for this probe. For more information on this ICMP types header, please refer to appendix ??, page ??.
- `icmp_echo_reply = y` in line 5 means that there was an ICMP echo request reply.
- `icmp_echo_code = !0` in line 6 means that the code in the ICMP header does not equal 0.
- In line 7, `icmp_echo_ip_id = !0` means that the identification number in the IP header does not equal 0.
- In line 8, `icmp_echo_tos_bits = !0` means that the type of service (`TOS`) or Differentiated Services Field in the IP header does not equal 0.
- `icmp_echo_df_bit = 1` in line 9 means that the don't fragment (`DF`) flag in the IP header is enabled.
- `icmp_echo_reply_ttl = <255` in line 10 means that the time to live (`TTL`) in the IP header is less than or equal to 255.

```
11 #Module B [ICMP Timestamp Probe]
12 icmp_timestamp_reply = y
13 icmp_timestamp_reply_ttl = <64
14 icmp_timestamp_reply_ip_id = !0
```

These lines represent the results from the ICMP timestamp test. For more information on this ICMP types header, please refer to appendix A.2, page 17.

- In line 12 `icmp_timestamp_reply = y` means that an ICMP timestamp reply message should be received from the target machine.
- In line 13 `icmp_timestamp_reply_ttl = <255` means that time to live in the IP header is less than or equal to 255.
- `icmp_timestamp_reply_ip_id = !0` in line 14 means that the identification number in the IP header does not equal 0.

```
15 #Module C [ICMP Address Mask Request Probe]
16 icmp_addrmask_reply = n
17 icmp_addrmask_reply_ttl = <64
18 icmp_addrmask_reply_ip_id = !0
```

These lines show the results from the ICMP Address Mask Request test. For more information on this ICMP types header, please refer to appendix ??, page ??.

- `icmp_addrmask_reply = n` in line 16 means that no reply should be received from the target device.
- Line 17 to 18 are ignored, since no reply is received from the target host for this probe type.

```
19 #Module D [ICMP Information Request Probe]
20 icmp_info_reply = n
21 icmp_info_reply_ttl = <64
22 icmp_info_reply_ip_id = !0
```

These lines show the results from the ICMP Information Request test. For more information on this ICMP types header, please refer to appendix A.4, page 18. Their meaning aligns with the parameters from Module C.

```
23 #Module E [UDP -> ICMP Unreachable probe]
24 #IP_Header_of_the_UDP_Port_Unreachable_error_message
25 icmp_unreach_reply = y
26 icmp_unreach_echoed_dtsize = 8
27 icmp_unreach_reply_ttl = <64
28 icmp_unreach_precedence_bits = 0
29 icmp_unreach_df_bit = 1
30 icmp_unreach_ip_id = !0
```

These lines show the results from the ICMP Port Unreachable test. For more information on this ICMP types header, please refer to appendix A.5, page 18.

- `icmp_unreach_echoed_dtsize = 8` in line 26 means that the amount of data returned with the ICMP port unreachable error message from the original UDP packet sent to the target machine is 8 bytes.
- `icmp_unreach_reply_ttl = <255` in line 27 means that the time to live in the IP header is less than or equal to 255.
- In line 28 the `icmp_unreach_precedence_bits = 0` means that the Precedence flag in the IP header is not enabled so it is equal to 0.
- `icmp_unreach_df_bit = 0` in line 29 means that the Don't Fragment flag in the IP header was not enabled so it is equal to 0.
- In line 30 `icmp_unreach_ip_id = !0` means that identification number in the IP header does not equal to 0.

Here is the example according to above in practice:

```
root@nostromo# xprobe2 nostromo.joeh.org

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

[+] Target is nostromo.joeh.org
[+] Loading modules.
[...]
[+] Host 193.170.32.26 Running OS: "FreeBSD 5.3" (Guess     ↲
  probability: 81%)
[...]
```

For a more in-depth explanation on ICMP based OS fingerprinting and `Xprobe2` see the PDF document written by Ofir Arkin, the author of `Xprobe2`, available on his homepage [Ark00].

Chapter 3

Defeating OS fingerprinting

In the effort to hide the running operating system of a device that is online, several strategies have emerged. Depending on the operating system some try to fool fingerprinting tools on kernel level, some on application level. Here are some tools described.

3.1 Linux Kernel Patches

3.1.1 IP Personality

`IP Personality`[1] is a patch for Linux kernels of version 2.4., that modifies the characteristics of network traffic. Things that can be influenced are: the TCP Initial Sequence Number, the TCP initial window size, the TCP options (their types, values and order in the packet), the IP ID numbers and answers to some pathological TCP packets. After applying this patch `iptables` has new targets that can be used in the mangle table.
It was written by Gaël Roualland and Jean-Marc Saffroy in 2001.

Since this patch is not maintained anymore experiments have been done with an old stable Debian release ("Woody") with the included alternative kernel, version 2.4.18 [2].
Here is the OS fingerprint before applying the patch:

```
root@nosint# nmap -F -sT -O 192.168.0.2
[...]
Running: Linux Kernel 2.4.0 - 2.5.20
[...]
```

Here is the OS fingerprint after having applied the patch, and having chosen to fake a MacOS-9 ("`samples/macos9.conf`").

[1]available at http://ippersonality.sourceforge.net/

[2]see http://www.debian.org/releases/woody/i386/release-notes/ch-whats-new.html for more details

```
root@nosint# nmap -F -sT -O 192.168.0.2
[...]
Running: Apple Mac OS 9, or HP-UX 11.00
[...]
```

`nmap` really got confused and returned a wrong guess.

[Rou01]

3.1.2 Stealth Patch

`Stealth Patch` from `Security Technologies` [3] is another patch for Linux kernels of version 2.2.19 through 2.2.22 and of version 2.4.19. When this patch is applied all packets with both FIN and SYN flag set [4] are discarded. Furthermore all packets with one specific reserved bit set [5] and all packets that match `nmaps` probe 2 – this means the ACK, FIN, RST and SYN flags are not set. Also packets with with the FIN, PUSH and URG flag set are dropped, which would equal to `nmaps` probe 7.
Though the `Stealth Patch` does not enable a host to fein being some other operating system, it still can confuse a fingerprinting application by droping specific packets that are typical for a OS detector (like `nmap` in this special case).
The downside of an unmodified `Stealth Patch` is, since only a few kernel versions are supported, this behaviour could give away valuable info to a fingerprinter again.
It was written by Sean Trifero & Derek Callaway in 2002.

3.1.3 Fingerprint Fucker

`Fingerprint Fucker` [6] is a kernel module available for Linux kernel of version 2.2. that also tries to hide the original OS and act as a different one. Per default it emulates the behaviour of a VAX device, but it can be configured by parsing a `nmap` signature file and hands over the values to the module. It mainly affects `nmap` tests one, two and seven.
It was written by |Cyrax| in 2000.

3.1.4 iplog

In contrary to the described methods up to now, `iplog` is not a kernel module but a standalone application. Although mainly written for detecting port

[3]available at http://www.innu.org/∼sean/

[4]This kind of probe is often called "QueSO probe", from the fingerprinting application `QueSO"`, available at http://www.l0t3k.net/tools/FingerPrinting/queso-980922.tar.gz

[5]These packets are called "bogus"

[6]http://www.s0ftpj.org/tools/fingfuck.tgz

scans it includes the ability to try to fool `nmap`. It detects TCP Null and FIN scans, UDP and ICMP "smurf" attacks, bogus TCP flags, TCP SYN and "Xmas" scans.
When starting this application the results were pretty disappointing. Maybe this application is far too old to still fool `nmap`, since it was written in 2001 and is not maintained anymore.
Here is the OS fingerprint before starting `iplog`:

```
root@nosint# nmap -F -sT -O 192.168.0.2
[...]
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.5.25 - 2.6.8 or Gentoo 1.2 Linux 2.4.19  ↲
  rc1-rc7
[...]
```

Here is the OS fingerprint after having started `iplog`:

```
root@nosint# nmap -F -sT -O 192.168.0.2
[...]
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.5.25 - 2.6.8 or Gentoo 1.2 Linux 2.4.19  ↲
  rc1-rc7, Linux 2.6.3 - 2.6.10
[...]
```

`iplog` did not really fool `nmap`, since only the string "`Linux 2.6.3 - 2.6.10`" was added. The operating system of the scanned host was a Debian Linux, running a kernel version 2.6.12.
It was written by Ryan McCabe in 2001.

[McC01]

3.2 FreeBSD, OpenBSD and NetBSD

3.2.1 blackhole

`blackhole` is an implementation for the `FreeBSD` kernel. If enabled the host does not return a RST segment if there is no socket accepting connections on a specific port. Instead it simply droppes the incoming packet and no RST segment is sent back. This makes the system look like a blackhole.
Here is the output of `Xprobe2` (as already known from chapter 2.3.2, page 8, here in a shorter form) before enabling `blackhole`:

```
[+] Target is nostromo.joeh.org
[+] Loading modules.
[...]
[+] Host 193.170.32.26 Running OS: "FreeBSD 5.3" (Guess       ↲
  probability: 81%)
[...]
```

Here is the output of `Xprobe2` after having enabled `blackhole`:

```
[+] Target is nostromo.joeh.org
[+] Loading modules.
[...]
[+] Host 193.170.32.26 Running OS: "Apple Mac OS X 10.3.7"    ↲
  (Guess probability: 100%)
[...]
[+] Execution completed.
```

`blackhole` fooled `Xprobe2` that good, it returned a 100% hit for the wrong guess "`Apple Mac OS X 10.3.7`".
It was written by Geoffrey M. Rehmet in 1999.

[Reh99]

3.2.2 Fingerprint Fucker

This is another application called `Fingerprint Fucker`, but this time it is for the `FreeBSD` operating system. It rewrites the TCP/IP stack and sends reply packets with differente settings like a different window size or TTL. It is available at http://packetstormsecurity.org/UNIX/misc/bsdfpf.tar.gz
It was written by cthulhu in 2001.

3.2.3 OpenBSD packet filter (`pf`)

`OpenBSD`'s built-in packet filter `pf` can also be configured to defeat OS fingerprinting techniques. Parameters like the Don't Fragment bit, the TTL, the Maximum Segment Size (MSS), the IP-ID field, etc. In the pf.conf man page [7] it is described as follows:

> no-df
>   Clears the don't-fragment bit from a matching ip packet.
> min-ttl _number_
>   Enforces a minimum ttl for matching ip packets.
> max-mss _number_
>   Enforces a maximum mss for matching tcp packets.

[7]http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf

random-id

Replaces the IP identification field with random values to compensate for predictable values generated by many hosts. This option only applies to outgoing packets that are not fragmented after the optional fragment reassembly.

3.2.4 FreeBSD TCP_DROP_SYNFIN

The `FreeBSD` kernel offered a special option, TCP_DROP_SYNFIN that dropped all packets with the FIN and SYN flags activated. This would prevent `nmap` test number 3 (see chapter 2.3.1 on page 6). Since there were serious troubles with `FreeBSD` hosts running a kernel with this option enabled and a webserver activated, it seems this option has been removed. Here is the text from an old archived handbook[8]:

> There are some other optional items that you can compile into the kernel for some added security. These are not required in order to get firewalling to work, but some more paranoid users may want to use them.
>
> options TCP_DROP_SYNFIN
>
> This option ignores TCP packets with SYN and FIN. This prevents tools like `security/nmap` from identifying the TCP/IP stack of the machine, but breaks support for RFC1644 extensions. This is not recommended if the machine will be running a web server.

[8]http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html

Appendix A

ICMP Headers

A.1 Echo Reply

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Identifier				Sequence Number			
Data							

Figure A.1: ICMP header fields, type 0 [Hun92]

A.2 Timestamp Request

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Identifier				Sequence Number			
Originate Timestamp							
Receive Timestamp							
Transmit Timestamp							

Figure A.2: ICMP header fields, Type 13 [Hun92]

A.3 Address Mask Request

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Identifier				Sequence Number			
Address Mask							

Figure A.3: ICMP header fields, Type 17 [Hun92]

A.4 Information Request

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Identifier				Sequence Number			

Figure A.4: ICMP header fields, Type 15 [Hun92]

A.5 Port Unreachable

4	8	12	16	20	24	28	32
Type		Code		Checksum			
Unused				Next Hop MTU			
IP header + the first 8 bytes of the original datagram's data.							

Figure A.5: ICMP header fields, Type 3 [Hun92]

Bibliography

[Ark00] Ofir Arkin. Icmp usage in scanning. URL, http://www.sys-security.com/archive/papers/ICMP_Scanning_v2.0.pdf, September 2000.

[Fyo98] Fyodor <fyodor@insecure.org>. Remote os detection via tcp/ip fingerprinting. URL, http://www.insecure.org/nmap/nmap-fingerprinting-article.html, October 1998.

[Fyo05] Fyodor <fyodor@insecure.org>. Who is fyodor? URL, http://www.insecure.org/myworld.html, April 2005.

[Hun92] Craig Hunt. *TCP/IP Network Administration*. O'Reilly, 1 edition, August 1992.

[JBB92] Van Jacobson, Bob Braden, and Dave Borman. Rfc 1323 - tcp extensions for high performance. URL, http://www.faqs.org/rfcs/rfc1323.html, May 1992.

[McC01] Ryan McCabe. iplog 2.2.3. URL, http://ojnk.sourceforge.net/stuff/iplog.readme, January 2001.

[Pos87] Jon Postel. Rfc 1025 - tcp and ip bake off. URL, http://www.faqs.org/rfcs/rfc1025.html, September 1987.

[Reh99] Geoffrey Rehmet. Blackhole(4). URL, http://www.freebsd.org/cgi/man.cgi?query=blackhole, August 1999.

[Rou01] Jean-Marc Roualland, Gaël Saffroy. Ip personality documentation. URL, http://ippersonality.sourceforge.net/doc/ippersonality-en.txt, July 2001.