A practical approach for defeating Nmap OS−Fingerprinting

David Barroso Berrueta

Table of Contents

A practical approach for defeating Nmap OS−Fingerprinting

Author: David Barroso Berrueta
http://voodoo.somoslopeor.com

Last updated: Mon Mar 10 19:23:32 CET 2003

Abstract
Remote OS Fingerprinting is becoming more and more important, not only for security pen−testers, but for the black−hat.
Just because Nmap is getting popularity as the tool for guessing which OS is running in a remote system, some security tools have been developed to fake Nmap in its OS Fingerprinting purpose.
This paper describes different solutions to defeat Nmap and behave like another chosen operating system, as well as a demonstration on how can be accomplished.

Table Index

Introduction

The purpose of this paper is to try to enumerate and briefly describe all applications and technics deployed for defeating Nmap OS Fingerprint, but in any case, security by obscurity is not good approach; it can be a good security measure but please take into account that is more important to have a tight security environment (patches, firewalls, ids, ...) than hiding your OS.

Learning which Operating System is running in a remote system can be very valuable for both the pen−tester and the black−hat. Suppose that they find an open port in their (approved or not) penetration; knowing the OS makes easier to find and execute an exploit against that service, because often an exploit is OS version specific, and an exploit for Sendmail running on HP−UX won't work for Sendmail running on AIX, or being more accurate, an AIX 4.3.3 exploit could not work in a system running 4.3.3 with the lastest maintenance code applied. Fyodor (Nmap's author) has written a detailed article about remote OS Fingerprint, describing some different methods to succesfully detect the remote OS, from the basic ones, to the more powerful ones.

In the beginning, guessing the remote OS was done grabbing the banner that a specific service was serving. For example, a typical telnet or ftp banner was always shown to the entire world, telling which OS was running, or if the banner has been changed or removed, some service commands could be executed to know the OS (remember the SYST in the FTP). Other basic ways to know the OS could be searching for HINFO entries in the DNS server, or trying to get information using snmp (lot of devices have enabled by default snmp access using the 'public' community string). Even searching for the target company jobs posting in the Internet, dumpster diving looking for OS manuals, or social engineering are valid methods for trying to know the remote OS.

Then, some more advanced network solutions were deployed, taking advantage of each OS vendor TCP/IP stack implementation. The idea is to send some crafted packets to the remote OS and wait for its answer. Those packets are "nasty" packets, crafted with uncommon TCP options or with 'impossible' options. Each OS has its own TCP/IP stack implementation, there isn't a common stack implementation for every OS and this issue allows to create a classification of different OS and versions according to their answers. Playing around with those tricky packets is how remote OS Fingerprinting tools work; some of them using the TCP/IP protocol, and others using the ICMP protocol.

There is a paper about 'Defeating TCP/IP Stack Fingerprinting' that describes in high level the design and implementation of a TCP/IP Stack fingerprint scrubber. That paper outlines why and how you can defeat TCP/IP OS Fingerprinting, so I am not going to talk too much about that; therefore I will focus on the solutions available out there.

Reasons to hide your OS to the entire world

Perhaps you are wondering why do you want to spend your precious time changing your linux kernel to hide your real OS version against Nmap 'bad purposes' users. Maybe the following reasons can convince you:

- Revealing your OS makes things easier to find and succesfully run an exploit against any of your devices.
- Having and unpatched or antique OS version is not very convenient for your company prestige. Imagine that your company is a bank and some users notice that you are running an unpatched box. They won't trust you any longer! In addition, these kind of 'bad' news are always sent to the public

opinion.

- Knowing your OS can also become more dangerous, because people can guess which applications are you running in that OS (data inference). For example if your system is a MS Windows, and you are running a database, it's highly likely that you are running MS−SQL.
- It could be convenient for other software companies, to offer you a new OS environment (because they know which you are running).
- And finally, privacy; nobody needs to know the systems you've got running.

Nmap

Nmap is one of such tools. It sends seven TCP/IP crafted packets (called tests) and waits for the answer. Results are checked against a database of known results (OS signatures database). This database is a text file that contains the result answered (signature) by each OS known. Thus, if the answer matches any of the entries in the database, we can guess that the remote OS is the same that the one in the database. Some Nmap packets are sent to an open port and the others to a closed port; depending on that results, the remote OS is guessed. A sample entry could be:

```
/* OS Comment. Yes, we want to be a Sega Dreamcast console */
Fingerprint Sega Dreamcast

/* ISN predictibilty; TD: time dependant */
TSeq(Class=TD%gcd=<780%SI=<14)

/* Test 1 result: SYN packet with some options to an open port. We got
a SYN+ACK, acknowledgment seq +1, window size 0x1d4c, don't fragment
bit not activated, and only the MSS returned */
T1(DF=N%W=1D4C%ACK=S++%Flags=AS%Ops=M)

/* Test 2 result: Null packet with options to an open port. We got a
ACK+RST, acknowledgment seq, window size 0x0, don't fragment bit not
activated */
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)

/* Test 3 result: SYN, FIN, URG, PSH with options to an open
port. We got a SYN+ACK, acknowledgment seq +1, window size 0x1d4c,
don't fragment bit not activated, and only the MSS returned */
T3(Resp=Y%DF=N%W=1D4C%ACK=S++%Flags=AS%Ops=M)

/* Test 4 result: ACK packet to an open port. We got a RST,
acknowledgment seq, window size 0x0, don't fragment bit not activated */
T4(DF=N%W=0%ACK=S%Flags=R%Ops=)

/* Test 5 result: SYN with options to a closed port. We got a
ACK+RST, acknowledgment seq, window size 0x0, don't fragment bit not
activated */
T5(DF=N%W=0%ACK=S%Flags=AR%Ops=)

/* Test 6 result: ACK with options to a closed port. We got a RST,
acknowledgment seq, window size 0x0, don't fragment bit not activated */
T6(DF=N%W=0%ACK=S%Flags=R%Ops=)

/* Test 7 result: FIN, PSH, URG with options to a closed port. We
got a ACK+RST, acknowledgment seq+q, window size 0x0, don't fragment
bit not activated */
```

```
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

/* Port unreachable message result. No response */
PU(Resp=N)
```

Then, if we want to defeat Nmap and tell the atacker that we are running a different operating system, we only need to fake the responses to the Nmap tests. The solution that is going to describe is only valid for defeating Nmap and not other remote OS Fingerprinting tools. In the Conclusion section, other tools will be mentioned, as well as some recomendations for the pen−tester and/or the attacker.

Linux solutions

Methods to defeat Nmap OS Fingerprinting in Linux are written as kernel modules, or at least, as patches to the linux kernel. The reason is that if the aim is to change Linux TCP/IP stack behaviour, and if we want to achieve it, we need to do it in the kernel layer.
Three kernel module solutions are going to be described, all of them independent from the linux kernel tree; you have to download them and patch your kernel to add the feature. The first one requires netfilter enabled in your kernel (what I think it's a must if you want to start to have a secure system), but the other two don't.

IP Personality

The first and probably, best option is IP Personality. It'a netfilter module (then, only available for 2.4 linux kernels) that allows us to change the IP stack behaviour and 'personality', having multiple network personalities depending on parameters that you can specify as an iptables rule. Actually, we can change the following options:

- TCP Initial Sequence Number (ISN)
- TCP initial window size
- TCP options (their types, values and order in the packet)
- IP ID numbers
- answers to some pathological TCP packets
- answers to some UDP packets

An IP Personality overall summary is that we can change the way we answer to some packets, and we can specify which packets we want to answer in such way (it could be depending on the source ip address, the destination port, or, and that's we are going to use, those crafted packets coming from Nmap).

Installation is fairly straight forward and well explained in the INSTALL file provided by the package; for our test purposes, our test box is a stable Debian box running a 2.4.19 kernel. By default, IP Personality netfilter module is not available in lastest kernel, so we need to patch our kernel sources. Patch for adding IP Personality feature to our netfilter core is available in the IP Personality site. We also need to patch the iptables command so that it can recognize our new feature available. Once the kernel is patched and compiled, we need to reboot our box just because the patch also modifies other netfilter files (the connection tracking).

Next step is include our iptables rules related to IP Personality in our working kernel. Before doing it, we run Nmap to check our current OS:

```
# nmap (V. 3.10ALPHA4) scan initiated Wed Feb 19 20:26:52 2003 as: nmap −sS −O −oN nmap1.log 19
Interesting ports on 192.168.0.19:
(The 1597 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
80/tcp     open        http
143/tcp    open        imap2
Remote operating system guess: Linux Kernel 2.4.0 − 2.5.20
Uptime 106.832 days (since Tue Nov  5 00:29:33 2002)

# Nmap run completed at Wed Feb 19 20:26:58 2003 −− 1 IP address (1 host up) scanned in 7.957 s
```

Now, we can reboot to run our new patched kernel and add the iptables rules needed to fake Nmap OS guess:

```
voodoo:~/ippersonality−20020819−2.4.19/samples#/usr/local/sbin/iptables −t mangle −A PREROUTING
voodoo:~/ippersonality−20020819−2.4.19/samples#/usr/local/sbin/iptables −t mangle −A OUTPUT −s
```

What we are doing with those filter rules is:

- The first one means that all packets coming from 192.168.0.50 (me) against 192.168.0.19 (server) have to be mangled and rewritten simulating a Dreamcast behaviour. The PREROUTING chain is the one that can do that.
- The second one means that all packets coming from 192.168.0.19 (server) against 192.168.0.50 (me) have to be mangled and rewritten simulating a Dreamcast behaviour. As is a packet going out the server, the OUTPUT chain is the responsible for that.

Checking our set−up:

```
voodoo:~/ippersonality−20020819−2.4.19/samples#/usr/local/sbin/iptables −L −t mangle
Chain PREROUTING (policy ACCEPT)
target     prot opt source            destination
PERS       all      192.168.0.50      192.168.0.19   tweak:dst local id:Dreamcast
Chain INPUT (policy ACCEPT)
target     prot opt source            destination
Chain FORWARD (policy ACCEPT)
target     prot opt source            destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source            destination
PERS       all      192.168.0.19      192.168.0.50   tweak:src local id:Dreamcast
Chain POSTROUTING (policy ACCEPT)
target     prot opt source            destination
```

It's time to see if Nmap can report that we are still runing a Linux kernel 2.4.0−2.5.20 or perhaps we can find out that our OS has changed:

```
# nmap (V. 3.10ALPHA4) scan initiated Wed Feb 19 21:49:18 2003 as: nmap −sS −O −oN nmap2.log 19
Interesting ports on 192.168.0.19:
(The 1597 ports scanned but not shown below are in state: closed)
Port       State       Service
```

```
22/tcp     open        ssh
25/tcp     open        smtp
80/tcp     open        http
143/tcp    open        imap2
Remote operating system guess: Sega Dreamcast

# Nmap run completed at Wed Feb 19 21:49:23 2003 −− 1 IP address (1 host up) scanned in 5.886 s
```

As you can see, we've fooled Nmap with our response. It's easy to choose the OS we want to 'run' in the Nmap OS fingerprint and tell IP Personality to behave like that chosen OS. Let's take a look to the dreamcast.conf file that we've specified when adding our iptables rules:

```
/* Our new OS identification */
id "Dreamcast";

/* only incoming packets will be mangled and TCP window sizes will not be changed*/
tcp {
  incoming yes;
  outgoing no;
  max−window 32768;
}

/* We need to emulate the Dreamcast ISN time dependant generator; this can be done with the fix
tcp_isn {
  type fixed−inc 2;
  initial−value random;
}

tcp_options {
  keep−unknown yes;
  keep−unused no;
  isolated−packets yes;
  code { copy(mss); }
}

/* now we have to follow nmap Dreamcast signature and answer like a Dreamcast */
tcp_decoy {
  code {
    if (option(mss)) { /* nmap has mss on all of its pkts */
      set(df, 0);
      if (listen) {
        if (flags(syn&ece)) { /* nmap test 1 */
          set(win, 0x1D4C);
          set(ack, this + 1);
          set(flags, ack|syn);
          insert(mss, this+1);
          reply;
        }
        if (flags(null)) { /* nmap test 2 */
          set(win, 0);
          set(ack, this);
          set(flags, ack|rst);
          reply;
        }
        if (flags(syn&fin&urg&push)) { /* nmap test 3 */
          set(win, 0x1D4C);
          set(ack, this + 1);
          set(flags, ack|syn);
          insert(mss, this+1);
```

```
          reply;
        }
        if (ack(0) && flags(ack) && !flags(syn|push|urg|rst)) { /* nmap test 4 */
          set(win, 0);
          set(ack, this);
          set(flags, rst);
          reply;
        }
      } else {
        set(win, 0);
        if (flags(syn) && !flags(ack)) { /* nmap test 5 */
          set(ack, this);
          set(flags, ack|rst);
          reply;
        }
        if (ack(0) && flags(ack) && !flags(syn|push|urg|rst)) { /* nmap test 6 */
          set(ack, this);
          set(flags, rst);
          reply;
        }
        if (flags(fin&push&urg)) { /* nmap test 7 */
          set(ack, this + 1);
          set(flags, ack|rst);
          reply;
        }
      }
    }
  }
}

/* No ICMP resonse for connections to closed UDP ports */
udp_unreach {
  reply no;
  df no;
  max−len 56;
  tos 0;

  mangle−original {
    ip−len 32;
    ip−id same;
    ip−csum zero;
    udp−len 308;
    udp−csum same;
    udp−data same;
  }
}
```

IP Personality is even more powerful. You can set up a linux firewall/router that will change the answer of the hosts behind it. All your hosts protected by that linux router can appear to be Sega Dreamcast consoles to any attacker!
There is also a great Nmap patch in the same site, named osdet, that allows us to OS Fingerprint an ip address (using Nmap engine), but with the fancy add−on that we can see the packets that are sent and their answer in tcpdump output format. Sometimes it's very helpful and easier to understand the OS Fingerprint technique watching the packets flowing on our screen (all Nmap tests and its answers).

Stealth patch

The solution that we are going to describe is the **stealth** patch, available from Security Technologies. It's available as a kernel modules for linux kernels 2.2.x (and in a near future for 2.4.x) ,and as a kernel patch (without module support for linux kernel 2.4.x). We're going to test the patch for the 2.4.19 kernel. When patched, two new options appear in our config file:

- IP: TCP Stack Options: option that we have to choose if want to use the stealth patch. If you select this option, it's enable by default when you boot your sistem. To disable them, you need to execute:

```
echo 0 > /proc/sys/net/ipv4/tcp_ignore_ack
echo 0 > /proc/sys/net/ipv4/tcp_ignore_bogus
echo 0 > /proc/sys/net/ipv4/tcp_ignore_synfin
```

- Log all dropped packets: logs all packets with bad options.

This patch simply discards the TCP/IP packets received with the following matches:

1. Packets with both SYN and FIN activated (tcp_ignore_synfin) (QueSO probe).
2. Bogus Packets: if the TCP header has the res1 bit active (one of the reserved bits, then it's a bogus packet) or it does not have any of the following activated: ACK, SYN, RST, FIN (Nmap test 2).
3. Packets with FIN, PUSH and URG activated (Nmap test 7).

This is a simpler solution than the one described earlier. We cannot behave like any other Operating System, we just silently drop all 'strange' packets that are supposed to be destinated to guess our OS, and hope that it will be enought to fool our attacker, or at least, make the things harder. These kernel modifications are easy to understand, and it would be relatively easy to add our own homemade 'bad packets' detection.

Fingerprint Fucker

Fingerprint Fucker is a kernel module available for linux kernel 2.2.x which also can hide your OS and behave like another. It's a kernel module which accepts parameters from the command line to configure the answer. By default, it simulates a VAX. There is also another file, called fing_parses.c, which parses a Nmap signature file and loads the Fingerprint Fucker module with the right parameters (when executing fing_parses, you have to specify which OS you want to emulate). It also waits for receiving a Nmap bogus packet, and then answers as you have configured. As far I've seen, only some Nmap tests are treated (T1, T2 and T7), and the code is not very stable. I loaded the module and in a few moments my linux box got frozen.

IPlog

IPlog is a TCP/IP logger that also detects some scans (XMAS, FIN, SYN, ...). For our purposes, it has an option (−z) that allows to fool Nmap queries, and, although we can behave as other OS, we can completely fool Nmap when guessing remotly our OS.

Now it's time to run iplog to check the results:

```
voodoo:~#iplog −o −L −z −i eth0
```

The options are the following: −o (don't fork and stay in foreground), −L (results to stdout), −z (fool Nmap),

−i eth0 (listen to eth0).If I run a Nmap against the box, iplog starts to write a lot of information to stdout, about all connections made, and even which type of scanning is being performed; I've included only the relevant information about Nmap OS Fingerprinting in the iplog's ouput:

```
Feb 20 13:20:54 TCP: SYN scan detected [ports 10082,1430,770,815,440,86,848,797,560,5998,...] f
Feb 20 13:20:56 TCP: Bogus TCP flags set by 192.168.0.50:49054 (dest port 22)
Feb 20 13:20:56 UDP: dgram to port 1 from 192.168.0.50:49047 (300 data bytes)
Feb 20 13:20:56 ICMP: 192.168.0.50: port is unreachable to (udp: dest port 1, source port 49047
Feb 20 13:20:58 UDP: dgram to port 1 from 192.168.0.50:49047 (300 data bytes)
Feb 20 13:20:58 ICMP: 192.168.0.50: port is unreachable to (udp: dest port 1, source port 49047
Feb 20 13:21:01 UDP: dgram to port 1 from 192.168.0.50:49047 (300 data bytes)
Feb 20 13:21:01 ICMP: 192.168.0.50: port is unreachable to (udp: dest port 1, source port 49047
Feb 20 13:21:04 TCP: Xmas scan detected [ports 1,9,49055,49056,49054] from 192.168.0.50 [ports
Feb 20 13:21:05 UDP: dgram to port 1 from 192.168.0.50:49047 (300 data bytes)
Feb 20 13:21:05 ICMP: 192.168.0.50: port is unreachable to (udp: dest port 1, source port 49047
Feb 20 13:21:12 TCP: null scan detected [ports 9,49056,49060,49054] from 192.168.0.50 [ports 49
Feb 20 13:21:13 TCP: FIN scan detected [ports 49060,49054,9,1] from 192.168.0.50 [ports 1,9,490
Feb 20 13:21:56 TCP: SYN scan mode expired for 192.168.0.50 − received a total of 1647 packets
Feb 20 13:21:56 TCP: Xmas scan mode expired for 192.168.0.50 − received a total
of 33812 packets (676300 bytes).
Feb 20 13:22:03 TCP: null scan mode expired for 192.168.0.50 − received a total
of 16462 packets (329300 bytes).
Feb 20 13:22:04 TCP: FIN scan mode expired for 192.168.0.50 − received a total of 16343 packets
```

Iplog does recognize the bogus TCP flags, null packet, ... every Nmap OS Fingerprint attempt. That's why it can act accordingly and send a fake answer to fool Nmap. Nmap output is the following:

```
# nmap (V. 3.10ALPHA4) scan initiated Thu Feb 20 13:20:54 2003 as: nmap −vv −sS
−O −oN nmap3.log 192.168.0.19
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Interesting ports on voodoo (127.0.0.1):
(The 1599 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
80/tcp     open        http
143/tcp    open        imap2
No exact OS matches for host (If you know what OS is running on it, see http://www.insecure.org
TCP/IP fingerprint:
SInfo(V=3.10ALPHA4%P=i586−pc−linux−gnu%D=2/20%Time=3E54C833%O=9%C=1)
T1(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=Y%DF=N%W=0%ACK=O%Flags=BA%Ops=)
T2(Resp=Y%DF=Y%W=100%ACK=O%Flags=BARF%Ops=)
T2(Resp=Y%DF=Y%W=100%ACK=O%Flags=BPF%Ops=)
T3(Resp=Y%DF=N%W=0%ACK=O%Flags=BA%Ops=)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=O%Flags=BA%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

# Nmap run completed at Thu Feb 20 13:21:07 2003 −− 1 IP address (1 host up) scanned in 13.633
```

As you can see, iplog answers to all the packets with specific options; we can have a look to iplog source code:

```
file iplog_tcp.c, line 99:

        if (opt_enabled(FOOL_NMAP) &&
                ((tcp_flags & TH_BOG) || (tcp_flags == TH_PUSH) || (tcp_flags == 0) ||
                ((tcp_flags & (TH_SYN | TH_FIN | TH_RST)) && (tcp_flags & TH_URG)) ||
                ((tcp_flags & TH_SYN) && (tcp_flags & (TH_FIN | TH_RST)))))
```

That 'if' statement means that if we have executed iplog with the '−z' switch (fool Nmap), and the TCP header options are:

- bogus (use of the reserved bits), or
- only PUSH , or
- NULL (no options), or
- SYN+URG, FIN+URG, RST+URG, or
- SYN+FIN, SYN+RST

then it will create a new packet for answering with the options we want (some options depend on the machine time, for example DF, that's why sometimes is 1 and other 0, or the window size which is defined as current_time & 1).
Of course we could change the file iplog_tcp.c so that iplog always behave as a Sega Dreamcast for those nasty packets, but we do not have the flexilibity to have multiple personalities or specify that we want to behave as a Dreamcast only for a specific traffic or ip address. It's a good idea to answer in this way to anormal packets, but it's better to have the control and be more granular.

*BSD solutions

Blackhole

Blackhole is a special option present in the *BSD kernel to control system behaviour when someone is connecting to closed TCP or UDP ports. There are two options that we can change:

```
sysctl −w net.inet.tcp.blackhole=[0 | 1 | 2]
sysctl −w net.inet.udp.blackhole=[0 | 1]
```

The TCP blackhole behaves as following: if the value is 0, whenever a packet connects a TCP closed port, it returns a RST. If the value is 1, if a SYN packet connects a TCP closed port, it's dropped; and if the value is 2, if any packet tries to connect to a TCP closed port, it's dropped.

The UDP blackhole is similar; if the value is 0, any connection to an UDP closed port, returns an ICMP port unreachable; if the value is 1, it does not return the ICMP port unreachable.

If we enable these settings in paranoid mode, tests 5, 6, 7 and the unreachable port test won't work when running Nmap to remotely guess the OS, so we'll not be able to know the OS.

Fingerprint fucker

There is also another Fingerprint fucker for the FreeBSD systems, written by Darren Reed, that simply rewrites the TCP/IP stack and sends packets with other settings (different window size, ttl, ...) trying to hide

its real OS.

OpenBSD packet filter

The OpenBSD packet filter can also be configured to try to defeat remote OS Fingerprint. There are some options in the ip.conf configuration file (Traffic Normalization) where you can change some IP fields (DF bit, TTL, MSS, ID), as you can see in ip.conf's man page:

```
no−df
      Clears the dont−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.

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

FreeBSD TCP_DROP_SYNFIN

FreeBSD kernel has got a special option, TCP_DROP_SYNFIN, which actually drops all packets with the SYN and FIN flags activated (Nmap test #3 sends a SYN+FIN+PSH+URG TCP packet); this special option could be also a valid method for defeating Nmap when performing its tests (be sure to activate it at startup in /etc/rc.conf).

General solutions

We saw when talking about IP Personality, that we could set up a linux router protecting our internal network, and that router could fool Nmap and other OS Fingerprinting tools when trying to remotely guess our internal network hosts' OS. If we haven't got a linux box, but we've got a Checkpoint FW−1, then we can do something similar because of the fw−1 INSPECT language. Using this language, it's easy to create your own 'packet inspector' for the packets that are going through your fw−1. There is a reference in the FW−1 mailing list describing a fw−1 service to manage those bogus packets:

```
Name: queSO
Match: tcp, (th_flags & TH_ACK, th_ack = 0) or (th_flags > 58) or (th_flags &
TH_SYN, th_flags & TH_FIN) or (th_flags & TH_RST, th_flags > TH_RST)
Prologue:
```

This is a queSO−specific service, but we can fool Nmap and other OS Fingerprinting tools by silently dropping the packets with this features. This solution only drops some packets, but does not change the window size, DF bit, or the mss. If you read carefully the mail sent to the FW−1 mailing list (and hosted in phoneboy FW−1 FAQ), it's also advised to change those settings in the boxes that are behind the fw−1, so that Nmap can be totally defeated.

More things to play with

Next solution won't allow us to hide or change our OS, but we'll be able to create as many virtual devices as we want with every valid Operating System you can imagine. This idea is being applied to the honeypots field, just because you can create a entire C class virtual network with lots of different OS flying around; the black−hat can be easily attracted by all those boxes running so many vulnerable services...It could be an attacker's heaven.

Honeypots in general, and this approach in particular, can be highly recommended not only for learning the black−hat tools and tactics, but for also divert attackers to your honeynet and not your production boxes. It can also make attackers think that you have an entire farm of a specific OS (the virtual one) and hide your real OS.

The package I'm going to briefly describe is honeyd, from Niels Provos. One of its greatest feature is that we can give each virtual device a specific OS personality. That personality is also fed by a standard nmap fingerprinting file, allowing us to become the OS we want. I'm not going to deeply describe this great tool, I'm only going to run the sample config file to demonstrate what it can do.

After installing it, there is a file which name is config.localhost with a lot of virtual devices configured in. For instance, if we get the device 10.0.0.1 definition:

```
route entry 10.0.0.1
route 10.0.0.1 link 10.0.0.0/24
[snip]
create routerone
set routerone personality "Cisco 7206 running IOS 11.1(24)"
set routerone default tcp action reset
add routerone tcp port 23 "router−telnet.pl"
[snip]
bind 10.0.0.1 routerone
[snip]
```

The high level explanation is that we have a device which ip address is 10.0.0.1, which will act as a Cisco 7206 running IOS 11.1(24), will reset all TCP connections except for connections to TCP port 23, because then the script router−telnet.pl (an emulation of the telnet daemon) will be executed. Well, let's run Nmap to check the OS running in the virtual device we've just created:

```
# nmap (V. 3.10ALPHA4) scan initiated Thu Feb 20 16:17:44 2003 as: nmap −v −sS −oN nmap4.log −O
Warning:  OS detection will be MUCH less reliable because we did not find at least 1 open and 1
Interesting ports on 10.0.0.1:
(The 1604 ports scanned but not shown below are in state: filtered)
Port       State       Service
23/tcp     open        telnet
Remote OS guesses: Cisco 7206 running IOS 11.1(24), Cisco 7206  (IOS 11.1(17)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=26314 (Worthy challenge)
IPID Sequence Generation: Incremental

# Nmap run completed at Thu Feb 20 16:20:42 2003 −− 1 IP address (1 host up) scanned in 178.847
```

Again, when receiving Nmap bogus packets, honeyd answers with the device's personality we've chosen.

Conclusion

As stated in the IP Personality Limitations, changing your TCP/IP stack behaviour when receiving Nmap bogus packets can create some troubles:

- some characteristics of OS are related to the host architecture (for instance page sizes on various CPU) which could lead to performance issues;
- some of these changes are more "political" choices of the IP stack (initial sequence numbers, window sizes, TCP options available...). Tweaking those allow to fool a scanner but might break regular connectivity by changing network parameters. It could also make the system weaker if the emulated IP stack is not as strong as the initial one.

In my opinion, it's pretty clear that we can't rely on only one secuity tool to remotely guess the Operating System. This paper has shown that it's very easy to fool Nmap (and other similar tools) when trying to profile a remote device, and that all those attempts can be properly logged by the remote administrator. To succesfully remotly fingerprint an OS, all possible methods have to be gathered, starting with the simpler ones (banner grabbing, seeking for job posts, social engineering, ...) to the more complex ones (network fingerprinting). Every open service in a remote device has to be properly analyzed (banner, responses, behaviour against attacks, DoS, known errors) and documented. It could be even possible (although not ethical) to run some tools that are known to crash specific OS versions (nuke, land, teardrop, ...) to clarify our guess.

Although all these solutions can be modified to detect and fool any other TCP/IP fingerprint tool (just knowing which packets are sent), it is highly recommended to use various tools when doing a remote OS Fingerprint. Nmap is perhaps the most widely used, but there is another tool that also works great: Xprobe. Xprobe also has got a signatures database (not updated very often), and the final guess it's a probabilistic guess (fuzzy matching) depending on various answers. One of xprobe's biggest problem is that it's rarely updated and it includes very few signatures. Nmap detects the remote OS if its tests' result is exactly equal to that OS signature in the database, but you can run Nmap with the switch (*−−osscan_guess or −−fuzzy*, and then it performs a more aggressive OS guess trying to find the best match available in its signatures database. There is a paper about Xprobe specification and usage where explains why its idea and implementation seems to be so good and so valid. I think it should be executed as a partner with Nmap, in case you can send both TCP and ICMP packets against the target host. Xprobe could be an effective tool in poorly secured networks, just because it sends ICMP timestamps and ICMP netmask requests, which can become suspicious for a network administrator. It does not sent bogus packets (uncommon TCP packets, since the reserved bits are rarely used) to detect the remote OS, it simply sends 'normal' traffic (ICMP) to the target host, making harder (if not impossible) to detect such packets (and therefore, act accordingly). This approach was first used in sing (Send Internet Nasty Garbage), which can be executed with the −O switch for doing OS Fingerprint (with the ICMP type you choose). It should be difficult to any IDS or network implementation to detect that those ICMP packets have other function, just because there are a huge number of those ICMP packets daily in our networks. On the other hand, ICMP now is getting blocked by default from almost every network environment, making impossible to do an ICMP OS remote fingerprint, but usually you can find some TCP services in those network environments and shoot your Nmap packets.

Just for being accurate, there is also another OS Fingerprint tool, named p0f; p0f listens to your network looking for the first SYN in a TCP connection and grabs that packet options. If it matches with its signature database, then we can guess the OS; again, changing any of the options that p0f is looking for, will completely fool it. If, for instance, using IP Personality, we change every packet's window size, we can fake our responses and fool p0f.

Administrators should also carefully configure all their devices for not showing anythong that can be used for

identified them (banners, issue, common services open by default, ...) and run one of these tools that can log the OS Fingerprint attempts, because it's very likely that, those ip addresses wanting to know your OS, will be attacking your network in a short period of time. Besides, setting up a linux router using IP Personality and fooling everyone outside your network that you're using a different OS (with any of the options shown in this paper), could be a good security measure.

References

Matthew Smart, G. Robert Malan, Farnam Jahanian. "Defeating TCP/IP Stack Fingerprinting". Usenix Security Symposium 2000
URL: http://www.usenix.org/publications/library/proceedings/sec2000/smart.html

Fyodor. "Nmap"
URL: http://www.insecure.org

Fyodor. "Remote OS Detection via TCP/IP Stack Fingerprinting". June 11, 2002
URL: http://www.insecure.org/nmap/nmap−fingerprinting−article.html

Gaël Roualland & Jean−Marc Saffroy. "IP Personality"
URL: http://ippersonality.sourceforge.net/

Gaël Roualland & Jean−Marc Saffroy. "Osdet Nmap patch"
URL: http://ippersonality.sourceforge.net/download.html

Sean Trifero & Derek Callaway. "Stealth"
URL: http://www.innu.org/~sean/

Ryan McCabe. "IPlog"
URL: http://ojnk.sourceforge.net/stuff/iplog.readme

Fusys & |CyRaX|. "Fingerprint Fucker"
URL: http://www.s0ftpj.org/tools/fingfuck.tgz

FreeBSD. "Blackhole"
URL: http://www.gsp.com/cgi−bin/man.cgi?section=4&topic=blackhole

Darren Reed. "Fingerprint Fucker"
URL: http://packetstormsecurity.org/UNIX/misc/bsdfpf.tar.gz

OpenBSD. "ip.conf manual"
URL: http://www.openbsd.org/cgi−bin/man.cgi?query=pf.conf&sektion=5&arch=i386&apr

FreeBSD. "Kernel Options"
URL: http://www.freebsd.org/doc/en_US.ISO8859−1/articles/dialup−firewall/kernel.html

Alfredo Andrés Omella. "Trying to stop the security tool queSO". FW1 Mailing list. October, 6th, 1998
URL: http://www.phoneboy.com/fom−serve/cache/82.html

Niels Provos. "Honeyd − Network Rhapsody for You"

URL: http://www.citi.umich.edu/u/provos/honeyd/

Gaël Roualland & Jean−Marc Saffroy. "IP Personality Limitations"
URL: http://ippersonality.sourceforge.net/doc/ippersonality−en−2.html

Fyodor Yarochkin & Ofir Arkin. "Xprobe"
URL: http://www.sys−security.com/html/projects/X.html

Fyodor Yarochkin & Ofir Arkin. "Xprobe2 − A'Fuzzy' Approach to Remote Active Operating System Fingerprinting". Version 1.0. August, 2nd, 2002
URL: http://www.sys−security.com/archive/papers/Xprobe2.pdf

Alfredo Andrés Omella. Sing
URL: http://sing.sourceforge.net

Michael Zalewski & William Stearns. "p0f"
URL: http://www.stearns.org/p0f/

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world−wide, royalty−free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front−matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front−Cover Texts or Back−Cover Texts, in the notice that says that the Document is released under this License. A Front−Cover Text may be at most 5 words, and a Back−Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine−readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard−conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine−generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a

specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front−Cover Texts on the front cover, and Back−Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine−readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer−network location from which the general network−using public has access to download using public−standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version

to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- ***A.*** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- ***B.*** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- ***C.*** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- ***D.*** Preserve all the copyright notices of the Document.
- ***E.*** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- ***F.*** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- ***G.*** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- ***H.*** Include an unaltered copy of this License.
- ***I.*** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- ***J.*** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- ***K.*** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- ***L.*** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- ***M.*** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- ***N.*** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- ***O.*** Preserve any Warranty Disclaimers.

If the Modified Version includes new front−matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties−−for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front−Cover Text, and a passage of up to 25 words as a Back−Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front−Cover Text and one of Back−Cover Text may be added by (or through arrangements made by) any one

entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.