# Remote OS detection via TCP/IP Stack FingerPrinting

by Fyodor <fyodor@insecure.org> (www.insecure.org)
Written: October 18, 1998
Last Modified: April 10, 1999

[French Translation by Arhuman <arhuman@francemel.com>]
[Portuguese Translation by Frank Ned <frank@absoluta.org>]
[Italian Translation by Rige <rigel@penguinpowered.com>]
[Russian Translation by Alex Volkov <topcat@cherepovets.ru>]
[Spanish Translation by Marco Barbosa <mabs@hotmail.com>]
[German Translation by Stefan Maly <stefan@maly.de>]
[Chinese Translation by neko <neko@126.com>]
[Turkish Translation by Egemen Tas <egement@karyde.com.tr>]
[Hebrew Translation by Elad <hax0r@netvision.net.il>]
[Japanese Translation by Yoriyuki Sakai <sakai@lac.co.jp>
and Hiromi Yanaoka <yanaoka@lac.co.jp>]



## *ABSTRACT*

This paper discusses how to glean precious information about a host by querying its TCP/IP stack. I first present some of the "classical" methods of determining host OS which do not involve stack fingerprinting. Then I describe the current "state of the art" in stack fingerprinting tools. Next comes a description of many techniques for causing the remote host to leak information about itself. Finally I detail my (nmap) implementation of this, followed by a snapshot gained from nmap which discloses what OS is running on many popular Internet sites.

## *REASONS*

I think the usefulness of determining what OS a system is running is pretty obvious, so I'll make this section short. One of the strongest examples of this usefulness is that many security holes are dependent on OS version. Lets say you are doing a penetration test and you find port 53 open. If this is a vulnerable version of Bind, you only get one chance to exploit it since a failed attempt will crash the daemon.

With a good TCP/IP fingerprinter, you will quickly find that this machine is running 'Solaris 2.51' or 'Linux 2.0.35' and you can adjust your shellcode accordingly. A worse possibility is someone scanning 500,000 hosts in advance to see what OS is running and what ports are open. Then when someone posts (say) a root hole in Sun's comsat daemon, our little cracker could grep his list for 'UDP/512' and 'Solaris 2.6' and he immediately has pages and pages of rootable boxes. It should be noted that this is SCRIPT KIDDIE behavior. You have demonstrated no skill and nobody is even remotely impressed that you were able to find some vulnerable .edu that had not patched the hole in time. Also, people will be even less impressed if you use your newfound access to deface the department's web site with a self-aggrandizing rant about how damn good you are and how stupid the sysadmins must be.

Another possible use is for social engineering. Lets say that you are scanning your target company and nmap reports a 'Datavoice TxPORT PRISM 3000 T1 CSU/DSU 6.22/2.06'. The hacker might now call up as 'Datavoice support' and discuss some issues about their PRISM 3000.

"We are going to announce a security hole soon, but first we want all our current customers to install the patch—I just mailed it to you ..." Some naive administrators might assume that only an

authorized engineer from Datavoice would know so much about their CSU/DSU. Another potential use of this capability is evaluation of companies you may want to do business with. Before you choose a new ISP, scan them and see what equipment is in use. Those "$99/year" deals don't sound nearly so good when you find out they have crappy routers and offer PPP services off a bunch of Windows boxes.

## *CLASSICAL TECHNIQUES*

Stack fingerprinting solves the problem of OS identification in a unique way. I think this technique holds the most promise, but there are currently many other solutions. Sadly, this is still one the most effective of those techniques:

```
playground~> telnet hpux.u-aizu.ac.jp
Trying 163.143.103.12 ...
Connected to hpux.u-aizu.ac.jp.
Escape character is '^]'.

HP-UX hpux B.10.01 A 9000/715 (ttyp2)
login:
```

There is no point going to all this trouble of fingerprinting if the machine will blatantly announce to the world exactly what it is running! Sadly, many vendors ship current systems with these kind of banners and many admins do not turn them off. Just because there are other ways to figure out what OS is running (such as fingerprinting), does not mean we should just announce our OS and architecture to every schmuck who tries to connect.

The problems with relying on this technique are that an increasing number of people are turning banners off, many systems don't give much information, and it is trivial for someone to "lie" in their banners. Nevertheless, banner reading is all you get for OS and OS Version checking if you spend $thousands on the commercial ISS scanner.

Download nmap or queso instead and save your money :). Even if you turn off the banners, many applications will happily give away this kind of information when asked. For example lets look at an FTP server:

```
payfonez> telnet ftp.netscape.com 21
Trying 207.200.74.26 ...
Connected to ftp.netscape.com.
Escape character is '^]'.
220 ftp29 FTP server (UNIX® System V Release 4.0) ready.
SYST
215 UNIX Type: L8 Version: SUNOS
```

First of all, it gives us system details in its default banner. Then if we give the 'SYST' command it happily feeds back even more information. If anon FTP is supported, we can often download /bin/ls or other binaries and determine what architecture it was built for. Many other applications are too free with information. Take web servers for example:

```
playground> echo 'GET / HTTP/1.0\n' | nc hotbot.com 80 | egrep '^Server:'
Server: Microsoft-IIS/4.0
playground>
```

Hmmm ... I wonder what OS those lamers are running. Other classic techniques include DNS host info records (rarely effective) and social engineering. If the machine is listening on 161/udp (SNMP), you are almost guaranteed a bunch of detailed info using 'snmpwalk' from the CMU SNMP tools distribution and the 'public' community name.

## *CURRENT FINGERPRINTING PROGRAMS*

Nmap is not the first OS recognition program to use TCP/IP fingerprinting. The common IRC spoofer sirc by Johan has included very rudimentary fingerprinting techniques since version 3 (or earlier). It attempts to place a host in the classes "Linux", "4.4BSD", "Win95", or "Unknown" using a few simple TCP flag tests.

Another such program is checkos, released publicly in January of this year by Shok in Confidence Remains High Issue #7. The fingerprinting techniques are exactly the same as SIRC, and even the code is identical in many places. Checkos was privately available for a long time prior to the public release, so I have no idea who swiped code from whom. But neither seems to credit the other. One thing checkos does add is telnet banner checking, which is useful but has the problems described earlier. [ Update: Shok wrote in to say that chekos was never intended to be public and this is why he didn't bother to credit SIRC for some of the code. ]

Su1d also wrote an OS checking program. His is called SS and as of Version 3.11 it can identify 12 different OS types. I am somewhat partial to this one since he credits my nmap program for some of the networking code :).

Then there is queso. This program is the newest and it is a huge leap forward from the other programs. Not only do they introduce a couple new tests, but they were the first (that I have seen) to move the OS fingerprints out of the code. The other scanners included code like:

```
/* from ss */
if ((flagsfour & TH_RST) && (flagsfour & TH_ACK) && (winfour == 0) &&
(flagsthree & TH_ACK))
reportos(argv[2],argv[3],"Livingston Portmaster ComOS");
```

Instead, queso moves this into a configuration file which obviously scales much better and makes adding an OS as easy as appending a few lines to a fingerprint file. Queso was written by Savage, one of the fine folks at Apostols.org .

One problem with all the programs describe above is that they are very limited in the number of fingerprinting tests which limits the granularity of answers. I want to know more than just 'this machine is OpenBSD, FreeBSD, or NetBSD', I wish to know exactly which of those it is as well as some idea of the release version number. In the same way, I would rather see 'Solaris 2.6' than simply 'Solaris'. To achieve this response granularity, I worked on a number of fingerprinting techniques which are described in the next section.

## *FINGERPRINTING METHODOLOGY*

There are many, many techniques which can be used to fingerprint networking stacks. Basically, you just look for things that differ among operating systems and write a probe for the difference. If you combine enough of these, you can narrow down the OS very tightly. For example nmap can reliably distinguish Solaris 2.4 vs. Solaris 2.5-2.51 vs Solaris 2.6. It can also tell Linux kernel 2.0.30 from 2.0.31-34 or 2.0.35. Here are some techniques:

**The FIN probe**—Here we send a FIN packet (or any packet without an ACK or SYN flag) to an open port and wait for a response. The correct RFC 793 behavior is to NOT respond, but many broken implementations such as MS Windows, BSDI, CISCO, HP/UX, MVS, and IRIX send a RESET back. Most current tools utilize this technique.

**The BOGUS flag probe**—Queso is the first scanner I have seen to use this clever test. The idea is to set an undefined TCP "flag" ( 64 or 128) in the TCP header of a SYN packet. Linux boxes prior to 2.0.35 keep the flag set in their response. I have not found any other OS to have this

bug. However, some operating systems seem to reset the connection when they get a SYN+BOGUS packet. This behavior could be useful in identifying them.

**TCP ISN Sampling**—The idea here is to find patterns in the initial sequence numbers chosen by TCP implementations when responding to a connection request. These can be categorized in to many groups such as the traditional 64K (many old UNIX boxes), Random increments (newer versions of Solaris, IRIX, FreeBSD, Digital UNIX, Cray, and many others), True "random" (Linux 2.0.*, OpenVMS, newer AIX, etc). Windows boxes (and a few others) use a "time dependent" model where the ISN is incremented by a small fixed amount each time period. Needless to say, this is almost as easily defeated as the old 64K behavior. Of course my favorite technique is "constant". The machines ALWAYS use the exact same ISN :). I've seen this on some 3Com hubs (uses 0x803) and Apple LaserWriter printers (uses 0xC7001).

You can also subclass groups such as random incremental by computing variances, greatest common divisors, and other functions on the set of sequence numbers and the differences between the numbers. It should be noted that ISN generation has important security implications. For more information on this, contact "security expert" Tsutomu "Shimmy" Shimomura at SDSC and ask him how he was owned. Nmap is the first program I have seen to use this for OS identification.

**Don't Fragment bit**— Many operating systems are starting to set the IP "Don't Fragment" bit on some of the packets they send. This gives various performance benefits (though it can also be annoying -- this is why nmap fragmentation scans do not work from Solaris boxes). In any case, not all OS's do this and some do it in different cases, so by paying attention to this bit we can glean even more information about the target OS. I haven't seen this one before either.

**TCP Initial Window**—This simply involves checking the window size on returned packets. Older scanners simply used a non-zero window on a RST packet to mean "BSD 4.4 derived". Newer scanners such as queso and nmap keep track of the exact window since it is actually pretty constant by OS type. This test actually gives us a lot of information, since some operating systems can be uniquely identified by the window alone (for example, AIX is the only OS I have seen which uses 0x3F25). In their "completely rewritten" TCP stack for NT5, Microsoft uses 0x402E. Interestingly, that is exactly the number used by OpenBSD and FreeBSD.

**ACK Value**—Although you would think this would be completely standard, implementations differ in what value they use for the ACK field in some cases. For example, lets say you send a FIN|PSH|URG to a closed TCP port. Most implementations will set the ACK to be the same as your initial sequence number, though Windows and some stupid printers will send your seq + 1. If you send a SYN|FIN|URG|PSH to an open port, Windows is very inconsistent. Sometimes it sends back your seq, other times it sends S++, and still other times is sends back a seemingly random value. One has to wonder what kind of code MS is writing that changes its mind like this.

**ICMP Error Message Quenching**—Some (smart) operating systems follow the RFC 1812 suggestion to limit the rate at which various error messages are sent. For example, the Linux kernel (in net/ipv4/icmp.h) limits destination unreachable message generation to 80 per 4 seconds, with a ¼ second penalty if that is exceeded. One way to test this is to send a bunch of packets to some random high UDP port and count the number of unreachables received. I have not seen this used before, and in fact I have not added this to nmap (except for use in UDP port scanning).

This test would make the OS detection take a bit longer since you need to send a bunch of packets and wait for them to return. Also dealing with the possibility of packets dropped on the network would be a pain.

**ICMP Message Quoting**—The RFCs specify that ICMP error messages quote some small amount of an ICMP message that causes various errors. For a port unreachable message,

almost all implementations send only the required IP header + 8 bytes back. However, Solaris sends back a bit more and Linux sends back even more than that. The beauty with this is it allows nmap to recognize Linux and Solaris hosts even if they don't have any ports listening.

**ICMP Error message echoing integrity**—I got this idea from something Theo De Raadt (lead OpenBSD developer) posted to comp.security.unix. As mentioned before, machines have to send back part of your original message along with a port unreachable error. Yet some machines tend to use your headers as 'scratch space' during initial processing and so they are a bit warped by the time you get them back. For example, AIX and BSDI send back an IP 'total length' field that is 20 bytes too high. Some BSDI, FreeBSD, OpenBSD, ULTRIX, and VAXen fuck up the IP ID that you sent them. While the checksum is going to change due to the changed TTL anyway, there are some machines (AIX, FreeBSD, etc.) which send back an inconsistent or 0 checksum. Same thing goes with the UDP checksum. All in all, nmap does nine different tests on the ICMP errors to sniff out subtle differences like these.

**Type of Service**—For the ICMP port unreachable messages I look at the type of service (TOS) value of the packet sent back. Almost all implementations use 0 for this ICMP error although Linux uses 0xC0. This does not indicate one of the standard TOS values, but instead is part of the unused (AFAIK) precedence field. I do not know why this is set, but if they change to 0 we will be able to keep identifying the old versions and we will be able to identify between old and new.

**Fragmentation Handling**—This is a favorite technique of Thomas H. Ptacek of Secure Networks, Inc (now owned by a bunch of Windows users at NAI). This takes advantage of the fact that different implementations often handle overlapping IP fragments differently. Some will overwrite the old portions with the new, and in other cases the old stuff has precedence. There are many different probes you can use to determine how the packet was reassembled. I did not add this capability since I know of no portable way to send IP fragments (in particular, it is a bitch on Solaris). For more information on overlapping fragments, you can read their IDS paper (www.secnet.com).

**TCP Options**—These are truly a gold mine in terms of leaking information. The beauty of these options is that:

1) They are generally optional (duh!) :) so not all hosts implement them.
2) You know if a host implements them by sending a query with an option set. The target generally show support of the option by setting it on the reply.
3) You can stuff a whole bunch of options on one packet to test everything at once.

Nmap sends these options along with almost every probe packet:
Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops; When you get your response, you take a look at which options were returned and thus are supported. Some operating systems such as recent FreeBSD boxes support all of the above, while others, such as Linux 2.0.X support very few. The latest Linux 2.1.x kernels do support all of the above. On the other hand, they are more vulnerable to TCP sequence prediction. Go figure.

Even if several operating systems support the same set of options, you can sometimes distinguish them by the values of the options. For example, if you send a small MSS value to a Linux box, it will generally echo that MSS back to you. Other hosts will give you different values. And even if you get the same set of supported options AND the same values, you can still differentiate via the order that the options are given, and where padding is applied. For example Solaris returns 'NNTNWME' which means:

<no op><no op><timestamp><no op><window scale><echoed MSS>

While Linux 2.1.122 returns MENNTNW. Same options, same values, but different order! I have not seen any other OS detection tools utilizes TCP options, but it is very useful. There are a few other useful options I might probe for at some point, such as those that support T/TCP and selective acknowledgements.

**Exploit Chronology**—Even with all the tests above, nmap is unable to distinguish between the TCP stacks of Win95, WinNT, or Win98. This is rather surprising, especially since Win98 came out about 4 years after Win95. You would think they would have bothered to improve the stack in some way (like supporting more TCP options) and so we would be able to detect the change and distinguish the operating systems. Unfortunately, this is not the case. The NT stack is apparently the same crappy stack they put into '95. And they didn't bother to upgrade it for '98.

But do not give up hope, for there is a solution. You can simply start with early Windows DOS attacks (Ping of Death, Winnuke, etc) and move up a little further to attacks such as Teardrop and Land. After each attack, ping them to see whether they have crashed. When you finally crash them, you will likely have narrowed what they are running down to one service pack or hotfix. I have not added this functionality to nmap, although I must admit it is very tempting :).

**SYN Flood Resistance**—Some operating systems will stop accepting new connections if you send too many forged SYN packets at them (forging the packets avoids trouble with your kernel resetting the connections). Many operating systems can only handle 8 packets. Recent Linux kernels (among other operating systems) allow various methods such as SYN cookies to prevent this from being a serious problem. Thus you can learn something about your target OS by sending 8 packets from a forged source to an open port and then testing whether you can establish a connection to that port yourself. This was not implemented in nmap since some people get upset when you SYN flood them. Even explaining that you were simply trying to determine what OS they are running might not help calm them.

## *NMAP IMPLEMENTATION AND RESULTS*

I have created a reference implementation of the OS detection techniques mentioned above (except those I said were excluded). I have added this to my Nmap scanner which has the advantage that it already knows what ports are open and closed for fingerprinting so you do not have to tell it. It is also portable among Linux, *BSD, and Solaris 2.51 and 2.6, and some other operating systems.

The new version of nmap reads a file filled with Fingerprint templates that follow a simple grammar. Here is an example:

```
FingerPrint  IRIX 6.2 - 6.4 # Thanks to Lamont Granquist
TSeq(Class=i800)
T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=C000|EF2A%ACK=O%Flags=A%Ops=NNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E
)
```

Lets look at the first line (I'm adding '>' quote markers):

```
> FingerPrint  IRIX 6.2 - 6.3 # Thanks to Lamont Granquist
```

This simply says that the fingerprint covers IRIX versions 6.2 through 6.3 and the comment states that Lamont Granquist kindly sent me the IP addresses or fingerprints of the IRIX boxes tested.

> TSeq(Class=i800)

This means that ISN sampling put it in the "i800 class". This means that each new sequence number is a multiple of 800 greater than the last one.

> T1(DF=N%W=C000|EF2A%ACK=S++%Flags=AS%Ops=MNWNNT)

The test is named T1 (for test1, clever eh?). In this test we send a SYN packet with a bunch of TCP options to an open port. DF=N means that the "Don't fragment" bit of the response must not be set. W=C000|EF2A means that the window advertisement we received must be 0xC000 or EF2A. ACK=S++ means the acknowledgement we receive must be our initial sequence number plus 1. Flags = AS means the ACK and SYN flags were sent in the response. Ops = MNWNNT means the options in the response must be (in this order):

<MSS (not echoed)><NOP><Window scale><NOP><NOP><Timestamp>
> T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)

Test 2 involves a NULL with the same options to an open port. Resp=Y means we must get a response. Ops= means that there must not be any options included in the response packet. If we took out '%Ops=' entirely then any options sent would match.

> T3(Resp=Y%DF=N%W=400%ACK=S++%Flags=AS%Ops=M)

Test 3 is a SYN|FIN|URG|PSH w/options to an open port.
> T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

This is an ACK to an open port. Note that we do not have a Resp= here. This means that lack of a response (such as the packet being dropped on the network or an evil firewall) will not disqualify a match as long as all the other tests match. We do this because virtually any OS will send a response, so a lack of response is generally an attribute of the network conditions and not the OS itself. We put the Resp tag in tests 2 and 3 because some operating systems do drop those without responding.

> T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
> T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
> T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)

These tests are a SYN, ACK, and FIN|PSH|URG, respectively, to a closed port. The same options as always are set. Of course this is all probably obvious given the descriptive names 'T5', 'T6', and 'T7' :).
>
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E
)

This big sucker is the 'port unreachable' message test. You should recognize the DF=N by now. TOS=0 means that IP type of service field was 0. The next two fields give the (hex) values of the IP total length field of the message IP header and the total length given in the IP header they are echoing back to us. RID=E means the RID value we got back in the copy of our original UDP packet was expected (ie the same as we sent). RIPCK=E means they didn't fuck up the checksum (if they did, it would say RIPCK=F). UCK=E means the UDP checksum is also correct. Next comes the UDP length which was 0x134 and DAT=E means they echoed our UDP data correctly. Since most implementations (including this one) do not send any of our UDP data back, they get DAT=E by default.

The version of nmap with this functionality is currently in the 6th private beta cycle. It may be out by the time you read this in Phrack. Then again, it might not. See http://www.insecure.org/nmap/

for the latest version.

## *POPULAR SITE SNAPSHOTS*

Here is the fun result of all our effort. We can now take random Internet sites and determine what OS they are using. A lot of these people have eliminated telnet banners, etc. to keep this information private. But this is of no use with our new fingerprinter! Also this is a good way to expose the <your favorite crap OS> users as the lamers that they are :)!
The command used in these examples was: nmap -sS -p 80 -O -v <host>
Also note that most of these scans were done on 10/18/98. Some of these folks may have upgraded/changed servers since then.

Note that I do not like every site on here.
# "Hacker" sites or (in a couple cases) sites that think they are
www.l0pht.com => OpenBSD 2.2 - 2.4
www.insecure.org => Linux 2.0.31-34
www.rhino9.ml.org => Windows 95/NT # No comment :)
www.technotronic.com => Linux 2.0.31-34
www.nmrc.org => FreeBSD 2.2.6 - 3.0
www.cultdeadcow.com => OpenBSD 2.2 - 2.4
www.kevinmitnick.com => Linux 2.0.31-34 # Free Kevin!
www.2600.com => FreeBSD 2.2.6 - 3.0 Beta
www.antionline.com => FreeBSD 2.2.6 - 3.0 Beta
www.rootshell.com => Linux 2.0.35 # Changed to OpenBSD after # they got owned.

# Security vendors, consultants, etc.
www.repsec.com => Linux 2.0.35
www.iss.net => Linux 2.0.31-34
www.checkpoint.com => Solaris 2.5 - 2.51
www.infowar.com => Win95/NT

# Vendor loyalty to their OS
www.li.org => Linux 2.0.35 # Linux International
www.redhat.com => Linux 2.0.31-34 # I wonder what distribution :)
www.debian.org => Linux 2.0.35
www.linux.org => Linux 2.1.122 - 2.1.126
www.sgi.com => IRIX 6.2 - 6.4
www.netbsd.org => NetBSD 1.3X
www.openbsd.org => Solaris 2.6 # Ahem :) (its because UAlberta
# is hosting them)
www.freebsd.org => FreeBSD 2.2.6-3.0 Beta

# Ivy league
www.harvard.edu => Solaris 2.6
www.yale.edu => Solaris 2.5 - 2.51
www.caltech.edu => SunOS 4.1.2-4.1.4 # Hello! This is the 90's :)
www.stanford.edu => Solaris 2.6
www.mit.edu => Solaris 2.5 - 2.51 # Coincidence that so many good
# schools seem to like Sun?
# Perhaps it is the 40%
# .edu discount :)
www.berkeley.edu => UNIX OSF1 V 4.0,4.0B,4.0D
www.oxford.edu => Linux 2.0.33-34 # Rock on!

# Lamer sites
www.aol.com => IRIX 6.2 - 6.4 # No wonder they are so insecure :)

www.happyhacker.org => OpenBSD 2.2-2.4 # Sick of being owned, Carolyn?
# Even the most secure OS is
# useless in the hands of an
# incompetent admin.

# Misc
www.lwn.net => Linux 2.0.31-34 # This Linux news site rocks!
www.slashdot.org => Linux 2.1.122 - 2.1.126
www.whitehouse.gov => IRIX 5.3
sunsite.unc.edu => Solaris 2.6

Notes: In their security white paper, Microsoft said about their lax security: "this assumption has changed over the years as Windows NT gains popularity largely because of its security features.". Hmm, from where I stand it doesn't look like Windows is very popular among the security community :). I only see 2 Windows boxes from the whole group, and Windows is easy for nmap to distinguish since it is so broken (standards wise).

And of course, there is one more site we must check. This is the web site of the ultra-secret Transmeta corporation. Interestingly the company was funded largely by Paul Allen of Microsoft, but it employs Linus Torvalds. So do they stick with Paul and run NT or do they side with the rebels and join the Linux revolution? Let us see:
We use the command:
nmap -sS -F -o transmeta.log -v -O www.transmeta.com//24
This says SYN scan for known ports (from /etc/services), log the
results to 'transmeta.log', be verbose about it, do an OS scan, and
scan the class 'C' where www.transmeta.com resides. Here is the gist
of the results:
neon-best.transmeta.com (206.184.214.10) => Linux 2.0.33-34
www.transmeta.com (206.184.214.11) => Linux 2.0.30
neosilicon.transmeta.com (206.184.214.14) => Linux 2.0.33-34
ssl.transmeta.com (206.184.214.15) => Linux unknown version
linux.kernel.org (206.184.214.34) => Linux 2.0.35
www.linuxbase.org (206.184.214.35) => Linux 2.0.35 ( possibly the same machine as above )
Well, I think this answers our question pretty clearly :).

## *ACKNOWLEDGEMENTS*

The only reason Nmap is currently able to detect so many different operating systems is that many people on the private beta team went to a lot of effort to search out new and exciting boxes to fingerprint! In particular, Jan Koum, van Hauser, Dmess0r, David O'Brien, James W. Abendschan, Solar Designer, Chris Wilson, Stuart Stock, Mea Culpa, Lamont Granquist, Dr. Who, Jordan Ritter, Brett Eldridge, and Pluvius sent in tons of IP addresses of wacky boxes and/or fingerprints of machines not reachable through the Internet.

Thanks to Richard Stallman for writing GNU Emacs. This article would not be so well word-wrapped if I was using vi or cat and ^D.

Questions and comments can be sent to fyodor@insecure.org. Nmap can be obtained from http://www.insecure.org/nmap .