Secure Coding.
Practical steps to defend your web apps.

OS and Application Fingerprinting Techniques

GSEC Gold Certification

Author: Jon Mark Allen, sans@allensonthe.net
Advisor: Carlos Cid, Ph.D.

Accepted: September 22, 2007

Outline

Introduction

This paper will attempt to describe what application and operating system (OS) fingerprinting are and discuss techniques and methods used by three of the most popular fingerprinting applications: nmap, Xprobe2, and p0f. I will discuss similarities and differences between not only active scanning and passive detection, but also the differences between the two active scanners as well. We will conclude with a brief discussion of why successful application or OS identification might be a bad thing for an administrator and offer suggestions to avoid successful detection.

Network scanning, and particularly remote OS/application detection, is generally the first step in mapping out a network; whether for penetration testing or simply maintaining a network device inventory. By understanding the methods network scanners use to determine remote OS/application versions, network administrators can better defend themselves against these types of reconnaissance probes. This paper will go beyond the simple listing of possible scan types such as TCP connect, SYN, and UDP scans, although those are certainly included. Instead, it will focus on the protocol stack details (sometimes known as TCP/IP stack *personalities*) and application quirks that allow these scanners to determine OS/application specifics.

1. What is an OS Fingerprint?

According to Wikipedia, "A [human] fingerprint is an impression of the friction ridges of all or any part of the finger" (*Fingerprint,* ¶1). Because no two fingerprints have ever been found

to be alike (German, ¶3), a fingerprint is an excellent tool to positively identify a person beyond reasonable doubt. Just like a fingerprint's unique pattern of ridge endings, bifurcations, and dots serve to identify an individual, an Operating System (OS) also has unique characteristics in its communication implementation that serve to identify it on a network. Those characteristics are also sometimes called a *personality*. By analyzing certain protocol flags, options, and data in the packets a device sends onto the network, we can make relatively accurate guesses about the OS that sent those packets. OS fingerprinting is the process of that analysis.

2. Why Does Fingerprinting Matter?

Why does remote fingerprinting matter? Wouldn't it be nice to run a scan on your own network and immediately know all the devices present and be able to tell the PCs from the servers from the routers from the firewalls? And after you had a baseline reference scan, you could run the same scan at regular intervals and know when a new device has been installed – without waiting for someone to tell you about it. Regular scanning with OS detection can help keep a network inventory clean and up to date.

On the other hand, if a *black hat* wanted to get into someone else's network, it would be incredibly useful to know what type of devices they have on the edge of their network. Armed with that knowledge, she can make intelligent guesses concerning the role devices play on the network and their importance. Typically, servers and firewalls are well fortified and monitored, but how many printers are kept up to date or even monitored at all? She can also eliminate

a tremendous number of possible vulnerabilities that don't exist on that platform. There's no need to try IIS exploits on an Apache server. And it's pointless to try to exploit Exchange vulnerabilities on a sendmail system. But if she finds a Cisco router, it's possible it hasn't been patched to fix the latest ICMP hole (US-CERT, 2007).

The first step of any intrusion will always be reconnaissance, and OS detection can be an excellent method for *black hats* to get to know your network.

3. Protocols and Fingerprinting

As practically everything on the Internet today, OS scanning works with the TCP/IP suite of protocols. In the real world and on the Internet, a protocol is simply an agreed upon set of rules used to communicate between two or more parties. Protocols are all around us and we use them all the time, usually without thinking about it. An example of a real world protocol is a telephone call. Before a conversation can take place, I first have to pick up my phone and dial your assigned number. When your phone rings, if you are available to talk, you would pick up the phone and say “Hello.” When I hear you answer, I would reply with “Hi [your name], this is Jon Mark” and hopefully you would answer with something like “Hey, Jon Mark” - after which our conversation would be established. If we are polite we would then take turns speaking and listening to the responses from the other end. We have protocols for driving cars, crossing the street, and any number of other daily activities. Protocols are part of our everyday life and protocols are what make

the Internet possible.

In the midst of the discussions of these protocols, I have included scaled down diagrams of each protocol header, as graciously provided by Matt Baxter, to help in understanding the layout and location of each of the fields. Each of these diagrams can be found in a larger format in Appendix A and are available in full size at the address listed in the Appendix.

TCP/IP

Computers have similar rules they must follow when communicating over a network, the most common of which is the TCP/IP suite of protocols. IP (Internet Protocol) is a method of assigning and managing logical addresses for each host on the network, while TCP (Transmission Control Protocol) ensures that all packets are delivered correctly. These protocols must be implemented in any operating system that wants to talk on the Internet. Both of these protocols are described in their respective RFCs, [791 and 793 for IP and TCP respectively] for developers to read and understand when

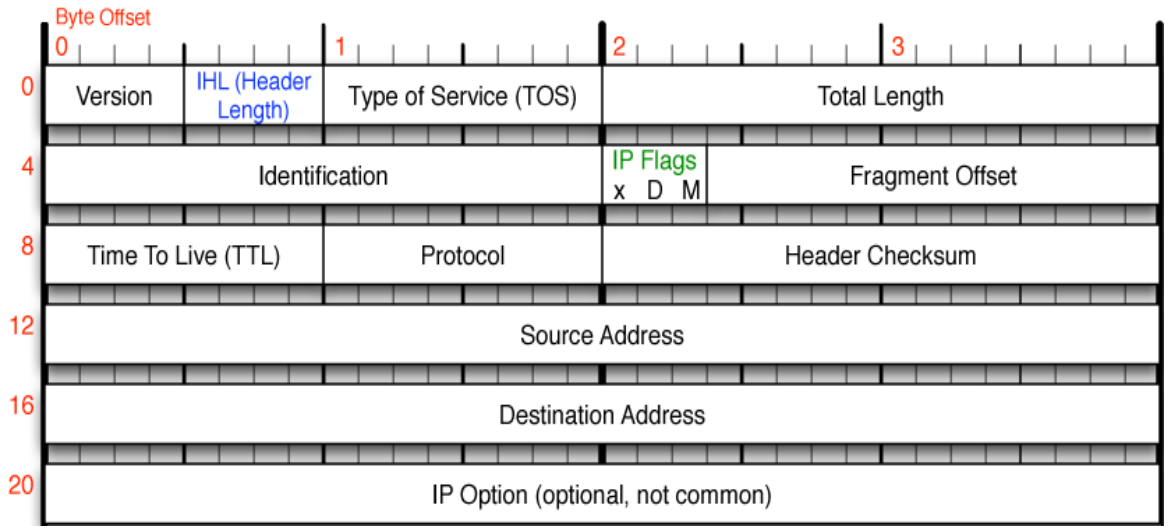

Figure 3.1.: IPv4 Header Diagram

implementing. However there are areas where the RFCs leave certain decisions to the developer.

Let's first examine the Time to Live (TTL) field of a TCP/IP packet. As we can see from Figure 3.1, this field is 8 bytes into the IP packet and defined on page 14, ¶2 of RFC 791:

> Time to Live: 8 bits
>
> This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Upon reading the above paragraph there is no doubt about how the TTL field should be used or how to modify it if the device is a router. But what should be the *initial* TTL value of a packet? For a maximum value, the developers are limited to the size of the field: 8 bits, which equates to a decimal value of 255. And you wouldn't want the value to be too low, because that would prohibit packets from reaching their destination even on a perfect network. But what *should* be used? It's really up to the development team to decide what they feel would be best for the operating system in development. In fact, after looking at a few packets on the wire from different hosts, we see that different OS's use different initial values. For example, Windows 2000 uses a value of 128, while Linux kernel 2.4 uses the value of 64. This information alone can help us make an intelligent guess about the OS of a host, if we have a fair idea of the (network) distance between devices.

Another field that proves to be interesting when fingerprinting systems is the Initial Sequence Number (ISN). The sequence number is

4 bytes into the TCP header as seen in Figure 3.2. Page 16, ¶1 of RFC 793 defines the Sequence Number field in a packet as:

> [the] sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

When using TCP, the two members of a conversation keep track of what data has been seen (and what data to expect next) by using sequence numbers. When first establishing a connection, each computer will select an ISN, and each subsequent packet will be numbered by counting up from that number, as shown in Figure 3.3 on the next page.

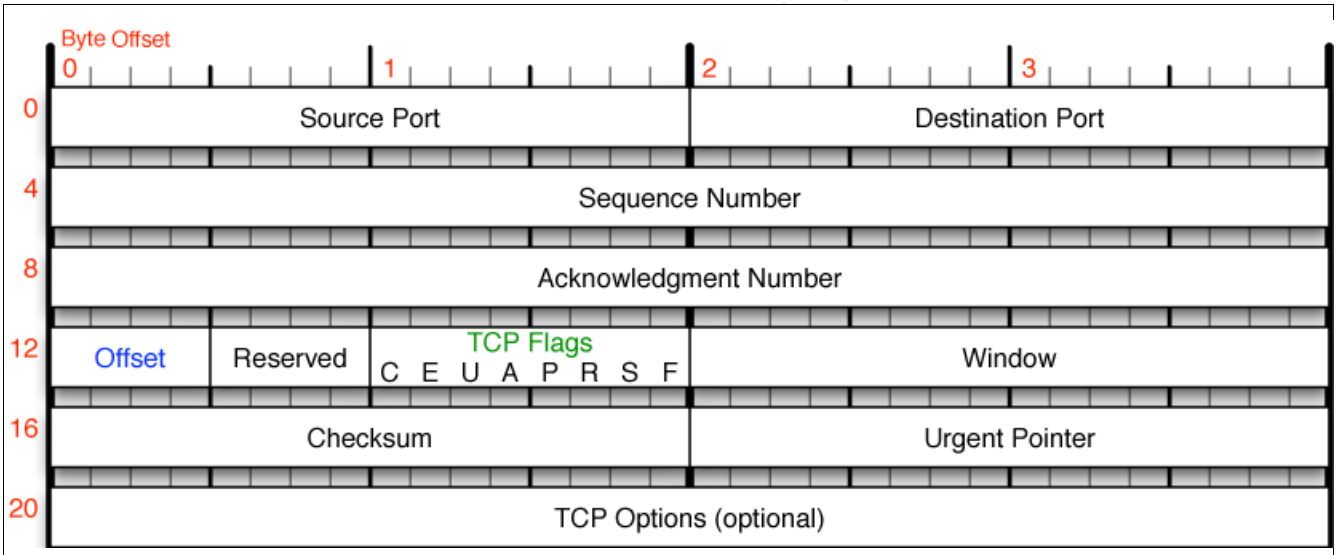

Figure 3.2.: TCP Header

An ISN was originally selected “by making use of a timed counter, like a clock of sorts, that was incremented every 4 microseconds. This counter was initialized when TCP started up and then its value increased by 1 every 4 microseconds until it reached the largest 32-bit value possible (4,294,967,295) at which point it 'wrapped around' to 0 and resumed incrementing.” (Kozierok, 2005) This method resulted in a rather serious security risk, since the value of an ISN could be guessed by an attacker and used to hijack a

connection. In 1996, RFC 1948 was proposed, which recommended taking the value of a combination of the internal clock, the 4-tuple of source IP and port plus the destination IP and port, and a pseudo-random number generator (PRNG) and passing that value through an MD5

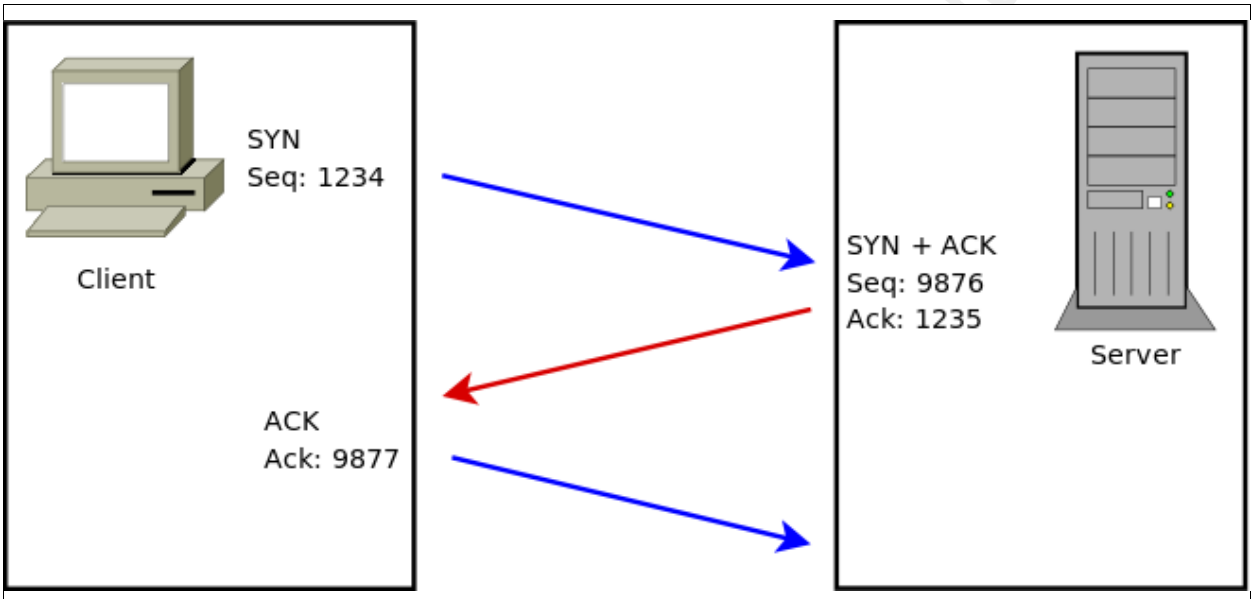

Figure 3.3.: TCP Three-Step Handshake with Sequence Numbers

hash function to create the ISN (1996, p. 3).

In 2001, and again in 2002, Michal Zalewski, the author of p0f, analyzed modern operating systems to analyze "relative network-based sequence number generators quality, which can be used to estimate attack feasibility and analyze underlying PRNG function behavior" (Zalewski, 2001)[1]. What he found was that most developers didn't follow the suggestions in RFC 1948. Usually, some variant of that combination (or at least a PRNG) was implemented. But not all PRNGs are created equal. By analyzing the ISNs generated by a target, including a test for consistent increment numbers, a scanner can possibly determine, or at least narrow down the possibilities of, the

1 Both of Mr Zalewski's papers are extremely fascinating and include compelling graphs of his findings. I consider them highly recommended reading.

target OS.

There are several TCP options which by themselves do not immediately reveal a target OS, but when examined together can significantly reduce the number of possibilities. These are Timestamps, Window Scaling, Maximum Segment Size, and Explicit Congestion Notification (ECN). Also appropriate for consideration is the IP Identification field. The following is a brief description of each of these fields.

The timestamp option was not part of the original TCP RFC, but was proposed in RFC 1072 and updated by RFCs 1185 and 1323. Timestamps were introduced to reduce the number of unnecessary retransmissions and therefore increase performance, particularly over long, high-speed links (RFC 1323, 1992, p. 1). The RFC defines the option as:

> The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option.
>
> The Timestamp Echo Reply field (TSecr) is only valid if the ACK bit is set in the TCP header; if it is valid, it echos a timestamp value that was sent by the remote TCP in the TSval field of a Timestamps option. When TSecr is not valid, its value must be zero. The TSecr value will generally be from the most recent Timestamp option that was received; however, there are exceptions that are explained [later in the RFC].

The combination of support for this option and the frequency with which the target updates the internal clock provide a value to consider in the OS decision matrix.

The TCP Window is a 16 bit unsigned field which defines the amount of data in bytes the receiver is ready to accept from the sender. As network bandwidth improved, to utilize the greater bandwidth it became prudent to be able to increase the window size

available for a host to advertise. Window scaling was proposed at the same time as timestamps and provides a value by which to multiply the window field, thus *scaling* the window while retaining compatibility with TCP stacks that don't support the option (RFC 1323).

The Maximum Segment Size (MSS) option was specified in the original TCP RFC and is used “to communicate the maximum receive segment size at the TCP which sends this segment” (RFC 793, 1981, p. 19). If this option is not present, any segment size is acceptable. Furthermore, this option should only be present in the SYN or SYN+ACK segments.

Historically, a host would realize there was network congestion when packets were dropped. Explicit Congestion Notification (ECN) provides a method for a network device (router or host) to inform other devices of congestion before packets are dropped. In this way, ECN-capable hosts can slow the rate of conversation and avoid packet loss. The diagram from RFC 3168 shows the ECN field as:

```
+-----+-----+
| ECN FIELD |
+-----+-----+
  ECT   CE        [Obsolete] RFC 2481 names for the ECN bits.
   0     0        Not-ECT
   0     1        ECT(1)
   1     0        ECT(0)
   1     1        CE
```

ECN-capable hosts will advertise their ability during the TCP three-step handshake by setting the ECT(0) or ECT(1) bit. Congestion is noted by setting both bits to 1 (RFC 3168, 2001, p. 7).

When a sending network is able to process larger packets than a receiving network, the datagrams are broken into multiple smaller

datagrams, called a fragment. The process as a whole is called fragmentation. To ensure reliable receipt, according to RFC 791,

> ...the internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed.
>
> The identification field is used to distinguish the fragments of one datagram from those of another.

The ID field is a 16 bit field in the IP header. Packets can also be tagged with a "Don't Fragment" (DF) bit, in which case the ID field is completely ignored (RFC 791, 1981, p. 8).

One interesting example of a fingerprint using the IP ID field is the Windows 95, 98, ME, and NT family of operating systems. This class of device increments the IP ID by a value of 256, instead of 1. Windows 200 and above reverted to the behavior adopted by all other OSs (Arkin, 2001, May 5).

The combination of support for these options and fields and their default values can all contribute to a host's TCP fingerprint, especially when taken in concert.

ICMP

Another protocol often used in fingerprinting is the Internet Control Message Protocol (ICMP). ICMP is defined in RFC 792 and is used when "a gateway or destination host [communicates] with a source host, for example, to report an error in datagram processing". All hosts that implement IP must also implement ICMP (RFC 792, 1981, p. 1). ICMP is a very useful protocol. It's also very simple, as can bee seen in Figure 3.4. Most traceroute utilities use ICMP to discover the network path a packet takes to its destination. Network

Administrators use ICMP daily to monitor the status of servers or routers, using a simple ICMP Echo (a.k.a. ping). And most Internet

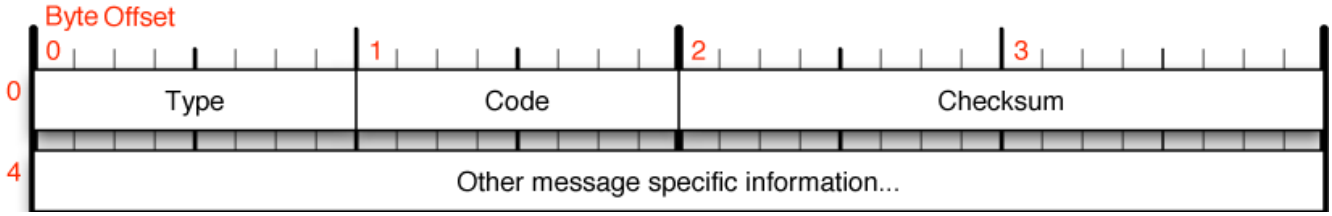

Figure 3.4.: ICMP Header Diagram

devices will respond to pings, but even a ping can reveal host OS information.

According to RFC 1349, Section 5.1, "An ICMP reply message is sent with the same value in the TOS field as was used in the corresponding ICMP request message." However, not all OS's correctly followed that directive (Arkin, Yarochkin, 2001, Aug 12, ¶6).

ICMP also returns error messages when a datagram is not processed correctly, whether due to the device not being active on the network or a problem with the datagram itself. These error messages can also be useful.

Page 9 of RFC 792 lists the description of ICMP type 12, "Parameter Problem Message", as:

> If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. ... The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

Furthermore, the device sending the error should include in the error message:

> the internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process.

RFC 792 was later modified in RFC 1122 to recommend including up

to 576 octets (or 4608 bits) in this error message (1989, Section 3.3.3, p. 60). However, most older TCP/IP implementations still include 8 octets, with some *nix flavors (and Mac) including more. And, as Arkin and Yarochkin point out, the development team for Solaris 2.x probably misread the RFC, as they included 64 octets (512 bits) instead of 64 bits (Arkin, Yarochkin, 2001, Aug 12).

Another notable fingerprint using ICMP and the IP ID field is the Linux 2.4.0 – 2.4.5 kernels. When replying to an ICMP Echo, these kernel versions set the DF bit (regardless of its setting on the echo packet) and the IP ID is set to zero. While this is completely legal (and on some levels, logical), it created an insanely obvious fingerprint which could be detected both actively and passively (Arkin, 2001, May 15).

Examples such as the ones above abound. At first glance, ICMP seems like an extremely simple protocol, but despite this appearance, it can reveal a tremendous amount of information about a router or host. For an exhaustive review (218 pages!) of ICMP behavioral differences, please refer to Ofir Arkin's book, *ICMP Usage in Scanning* (Arkin, 2001, June).

4. Scanners

In general, there are two types of scanners: active and passive. Active scanners work by sending a series of specially crafted packets to the target host and analyzing the replies. This allows the scanner to obtain more accurate results than a passive scanner and in a shorter amount of time. However, if the target host or network is utilizing an IDS, it also allows the scanner's actions to be

potentially detected and blocked. There is also some debate as to the legal nature of actively scanning a network (Lasser, ¶6). Passive scanners, on the other hand, do not generate any traffic during the fingerprinting session, but instead analyze existing traffic between the scanning host and the target. While the legal nature of an active scan might be in question, there is no doubt about passive fingerprinting. A passive scanner is simply analyzing traffic that has already been sent. As one writer put it, “it's like listening to other people's accent in a cafe” (ibid.). A passive scanner can also be an independent third-party device which receives a “copy” of the traffic to be analyzed. In fact, passive fingerprinting can be done completely offline, by examining packets captured previously. Passive scanners are generally and inherently less accurate than active scanners, due to the fact they have less control over the data they are analyzing. In the p0f README file, even the author suggests the ideal situation for his tool is when nmap (or Xprobe2) won't work (Zalewski, 2006, Section 2 ¶6). On the other hand, passive fingerprinting is almost completely undetectable.

nmap

“nmap ('Network Mapper') is an open source utility for network exploration or security auditing.” (nmap, Introduction ¶1) nmap is an active scanner and arguably one of the most popular network scanners in use today. nmap was introduced to the Internet community by its original author, Fyodor, in issue 51 of Phrack (1997). The primary purpose of nmap is to rapidly scan large segments of a network for devices that are active, with the ability to report which ports are open on those devices.

nmap has possessed OS detection abilities for some time, but the latest 4.x version of code has been re-written and much improved (Biancuzzi, 2001). nmap utilizes a text-based database of over 1500 entries accumulated over the years and contributed by the community. Keep in mind, nmap is primarily a network scanner, which includes an OS detection module. This affects the design of the software and the goals of the project. If stealth operation is of paramount importance to you, nmap might not be the tool you want to use. That's not to say nmap can't be used in a quieter mode, but if you use the defaults, you are more likely to be noticed by the target.

Since this paper is not intended to reproduce the documentation for nmap, but to give a solid understanding of OS detection, not all tests will be covered as they are too numerous to be given proper attention. We will, however, examine some of the most basic tests and review how they are calculated. We will also show the output of the scan as seen by tcpdump, which I believe helps to understand exactly what is going on behind the scenes. The tests are broken up into two main sections: probes sent and responses received.

"nmap OS fingerprinting works by sending up to 15 TCP, UDP, and ICMP and probes to known open and closed ports of the target machine." (*TCP/IP Fingerprinting Methods Supported by* nmap, ¶1) Probe packets are sent to at least one open and one closed port, therefore OS detection can only be performed after a port scan has been completed[1]. Both the port scan and OS detection are completed with only one command from the user. The probe tests we will examine

1 Which means the actual number of packets sent will always be greater than 15, but it is possible to limit the ports scanned and thus still keep the sent packet count low.

relate to TCP ISN generation, ICMP echo, TCP ECN, and six single TCP packets with variable options, and a UDP packet to a closed port.

Our examined response tests, which are carried out on the replies to the probe packets, will revolve around ISN, IP ID metrics, timestamps, and TCP options. These tests are probably best understood by examining them via a packet capture.

First, we will look at the nmap command that generated our packet capture and the results nmap displayed on the console.

```
jm@linuxbox> sudo nmap -sV -T5 -P0 -O 10.10.2.102
Starting nmap 4.20 ( http://insecure.org ) at 2007-04-14 11:07 CDT
Warning: Giving up on port early because retransmission cap hit.
Interesting ports on 10.10.2.102:
Not shown: 1111 closed ports, 585 filtered ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 4.5 (protocol 2.0)
MAC Address: 00:0A:95:92:F5:1E (Apple Computer)
Device type: general purpose
Running: Apple Mac OS X 10.4.X
OS details: Apple Mac OS X 10.4.8 (Tiger)
Network Distance: 1 hop
OS and Service detection performed. Please report any incorrect results at
http://insecure.org/nmap/submit/ .
nmap finished: 1 IP address (1 host up) scanned in 11.124 seconds
```

You will note that anything other than the default TCP Connect scan requires root privileges and since you should only run as root when absolutely necessary, we've used sudo to temporarily gain root privileges for this command only.

The `-sV` option tells nmap to attempt to determine the actual service (and the version where possible) running on the port. Without this option, nmap simply checks the `/etc/services` file and reports the service listed there for the open port. But there's nothing that says you can't run a web server on port 22 instead of an SSH server. Running services on non-standard ports can be a form of security through obscurity, a practice that is sometimes debated but generally

looked down upon (Perens, 1998). This option proves it is unwise to depend on utilizing non-standard ports for security.

The `-T` option specifies how fast nmap should scan the host, with a value between 0 and 5, with a higher number indicating a faster scan. There are several reasons a scanner might want to operate slowly. A slower scan reduces the risk of target detection. It also minimizes the possibility of flooding the target and creating a denial of service, especially if the target has a fragile TCP stack, such as some Old HP Jet Direct cards (*ISS Security Advisory*, 1998).

The `-P0` option tells nmap not to send an ICMP Echo before beginning the port scan. And the `-O` option tells nmap to perform remote OS detection, which is our focal point.

During the course of the port scan, nmap determines TCP port 22 to be open on the target host, as shown by our tcpdump capture:

```
11:07:13.140277 IP 10.10.2.103.45977 > 10.10.2.102.ssh: S 2310181754:2310181754(0) win 2048
<mss 1460>
11:07:13.140540 IP 10.10.2.102.ssh > 10.10.2.103.45977: S 1690505229:1690505229(0) ack
2310181755 win 65535 <mss 1460>
11:07:13.140597 IP 10.10.2.103.45977 > 10.10.2.102.ssh: R 2310181755:2310181755(0) win 0
```

After the scanner receives a SYN/ACK from the target, it then sends a RST packet to close the connection. The next step is to determine the service (and version if possible) running on the port. This time, the scanner completes the TCP three-step handshake and the host responds with:

```
11:07:19.531234 IP 10.10.2.102.ssh > 10.10.2.103.35979: P 1:21(20) ack 1 win
65535 <nop,nop,timestamp 257823113 1790405>
        0x0000:  4500 0048 768c 4000 4006 ab43 0a0a 0266  E..Hv.@.@..C...f
        0x0010:  0a0a 0267 0016 8c8b 4ced 2fb8 e8ff 61fc  ...g....L./...a.
        0x0020:  8018 ffff df10 0000 0101 080a 0f5e 1189  .............^..
        0x0030:  001b 51c5 5353 482d 322e 302d 4f70 656e  ..Q.SSH-2.0-Open
        0x0040:  5353 485f 342e 350a                      SSH_4.5.
```

We can plainly see in the ASCII portion of the packet the protocol version and the version of the server running the service. This sort of detection is called banner grabbing. Sometimes, application banners such as this produce even more information:

```
15:04:17.978365 IP 10.10.10.37.ssh > 10.10.2.103.47425: P 1:39(38) ack 1 win
1024 <nop,nop,timestamp 524259 524257>
        0x0000:  4500 005a 4ef2 4000 4006 eda9 0a0a 0a25  E..ZN.@.@.......
        0x0010:  0a0a 0267 0016 b941 3625 b113 360a 9e6c  ...f...A6%..6..l
        0x0020:  8018 0400 fe4e 0000 0101 080a 0007 ffe3  .....N..........
        0x0030:  0007 ffe1 5353 482d 322e 302d 4f70 656e  ....SSH-2.0-Open
        0x0040:  5353 485f 342e 3370 3220 4465 6269 616e  SSH_4.3p2.Debian
        0x0050:  2d38 7562 756e 7475 310a                 -8ubuntu1.
```

Not only can we see the version of SSH on the target, but the distribution, too! This sort of information is never necessary in a banner. Protocol versions can be helpful and sometimes necessary, but it is certainly not prudent to advertise unnecessary information such as the distribution.

Other common services that are particularly prone to banner grabbing are web servers:

```
12:59:53.388643 IP scanme.nmap.org.www > 10.10.10.77.48818: P 1:386(385) ack 109 win 46
<nop,nop,timestamp 2573553875 1062783>
        0x0000:  4500 01b5 ed89 4000 3106 df4a cdd9 993e  E.....@.1..J...>
        0x0010:  0a0a 0a4d 0050 beb2 2fd7 dbcb f376 c45d  ...M.P../....v.]
        0x0020:  8018 002e bb15 0000 0101 080a 9965 50d3  .............eP.
        0x0030:  0010 377f 4854 5450 2f31 2e31 2034 3031  ..7.HTTP/1.1.401
        0x0040:  2041 7574 686f 7269 7a61 7469 6f6e 2052  .Authorization.R
        0x0050:  6571 7569 7265 640d 0a44 6174 653a 2053  equired..Date:.S
        0x0060:  6174 2c20 3235 2041 7567 2032 3030 3720  at,.25.Aug.2007.
        0x0070:  3137 3a35 393a 3438 2047 4d54 0d0a 5365  17:59:48.GMT..Se
        0x0080:  7276 6572 3a20 4170 6163 6865 2f32 2e32  rver:.Apache/2.2
        0x0090:  2e32 2028 4665 646f 7261 290d 0a57 5757  .2.(Fedora)..WWW
        0x00a0:  2d41 7574 6865 6e74 6963 6174 653a 2042  -Authenticate:.B
        0x00b0:  6173 6963 2072 6561 6c6d 3d22 4e6d 6170  asic.realm="Nmap
        0x00c0:  2d57 7269 7465 7273 2043 6f6e 7465 6e74  -Writers.Content
        [output trimmed]
```

and SMTP servers:[1]

```
13:14:27.919531 IP mail5.abcde.com.smtp > 10.10.10.77.56528: P 728834299:728834432(133) ack
722304377 win 65535 <nop,nop,timestamp 8248469 1281339>
        0x0000:  4500 00b9 2c32 4000 7006 824f 0101 0101  E...,2@.p..O....
        0x0010:  0a0a 0a4d 0019 dcd0 2b71 20fb 2b0d 7d79  ...M....+q..+.}y
        0x0020:  8018 ffff ea91 0000 0101 080a 007d dc95  .............}..
        0x0030:  0013 8d3b 3232 3020 4142 4344 452d 4558  ...;220.ABCDE-EX
        0x0040:  494d 4332 2e65 7863 6861 6e67 652e 6162  IMC2.exchange.ab
        0x0050:  6364 652e 636f 6d20 4d69 6372 6f73 6f66  cde.com.Microsof
        0x0060:  7420 4553 4d54 5020 4d41 494c 2053 6572  t.ESMTP.MAIL.Ser
        0x0070:  7669 6365 2c20 5665 7273 696f 6e3a 2036  vice,.Version:.6
        0x0080:  2e30 2e33 3739 302e 3339 3539 2072 6561  .0.3790.3959.rea
        0x0090:  6479 2061 7420 2053 6174 2c20 3235 2041  dy.at..Sat,.25.A
        0x00a0:  7567 2032 3030 3720 3134 3a31 343a 3237  ug.2007.14:14:27
        0x00b0:  202d 3034 3030 200d 0a                   .-0400...
```

nmap displays these services as:

```
jm@linuxbox> sudo nmap -sV -P0 -p 80 scanme.insecure.org

Starting Nmap 4.20 ( http://insecure.org ) at 2007-08-25 12:59 CDT
Interesting ports on scanme.nmap.org (205.217.153.62):
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.2 ((Fedora))

jm@linuxbox> sudo nmap -sV -P0 -p 25 mail5.abcde.com

Starting Nmap 4.20 ( http://insecure.org ) at 2007-08-25 13:14 CDT
Interesting ports on mail5.abcde.com (1.1.1.1):
PORT   STATE SERVICE VERSION
25/tcp open  smtp    Microsoft ESMTP 6.0.3790.3959
Service Info: Host: ABCDE-EXIMC2.exchange.abcde.com; OS: Windows
```

Now it's finally time to begin the OS detection probe packets. The first test covers TCP sequence generation and consists of a series of six packets sent to a known open port. Timing is a critical factor in calculating the ISN, IP ID, and timestamp calculations, which depend on exact timing for accuracy. Therefore, nmap attempts to send each packet exactly 110 milliseconds apart, for a total time of 550ms. nmap's documentation lists the six probe packets as follows (MSS refers to the Maximum Segment Size value):

- Packet #1: Window scale (10), NOP, MSS (1460). Window field: 1.
- Packet #2: MSS (1400), Window scale (0). Window field: 63.

1 The SMTP packet header has been modified to protect the innocent.

- Packet #3: NOP, NOP, Window scale (5), NOP, MSS (640). Window field: 4.
- Packet #4: Window scale (10). Window field: 4.
- Packet #5: MSS (536), Window scale (10). Window field: 16.
- Packet #6: MSS (265). Window field: 512.

All packets have a timestamp value (TSval) of 0xFFFFFF and timestamp echo reply (TSecr) of 0. All packets except #3 have Selective ACK (SACK) permitted. (*TCP/IP Fingerprinting Methods Supported by nmap*, Probes Sent, ¶4)

We can see these packets in our packet capture (shown here minus the resulting SYN+ACK and RST packets).

```
21:09:30.259236 IP 10.10.2.105.44907 > 10.10.2.102.22: S 3752614626:3752614626(0) win 1
<wscale 10,nop,mss 1460,timestamp 4294967295 0,sackOK>
21:09:30.363638 IP 10.10.2.105.44908 > 10.10.2.102.22: S 3752614627:3752614627(0) win 63 <mss
1400,wscale 0,sackOK,timestamp 4294967295 0,eol>
21:09:30.467233 IP 10.10.2.105.44909 > 10.10.2.102.22: S 3752614628:3752614628(0) win 4
<timestamp 4294967295 0,nop,nop,wscale 5,nop,mss 640>
21:09:30.571334 IP 10.10.2.105.44910 > 10.10.2.102.22: S 3752614629:3752614629(0) win 4
<sackOK,timestamp 4294967295 0,wscale 10,eol>
21:09:30.675254 IP 10.10.2.105.44911 > 10.10.2.102.22: S 3752614630:3752614630(0) win 16 <mss
536,sackOK,timestamp 4294967295 0,wscale 10,eol>
21:09:30.779253 IP 10.10.2.105.44912 > 10.10.2.102.22: S 3752614631:3752614631(0) win 512
<mss 265,sackOK,timestamp 4294967295 0>
```

The reply packets are processed through a large variety of additional tests, the full description of which are listed in the nmap documentation (*TCP/IP Fingerprinting Methods Supported by nmap*).

There are three tests around the ISN: Greatest Common Denominator (GCD), sequence counter rate, and sequence predictability index. The GCD of the ISNs in the response packets is calculated in "an attempt to determine the smallest number by which the target host increments the [ISN]" (*TCP/IP Fingerprinting Methods Supported by nmap*, Response Tests, ¶2). The counter rate test "reports the average rate of increase for the returned TCP [ISN]" (ibid., ¶4). The sequence predictability test "measures the average rate of [ISN]

increments" and "roughly estimates how difficult it would be to predict the next ISN from the known sequence of six probe responses" (ibid., ¶5).

The IP ID based tests are also threefold and focus on sequence analysis: TCP sequence generation, ICMP sequence generation, and a shared sequence indicator. TCP sequence generation "examines the IP header ID field for every response to the TCP probes" and classifies the target generator in one of seven categories: all zero, random, all equal, random positive incremental, broken incremental, incremental, or unclassified (ibid, ¶8). "Broken incremental" describes the condition when the IP ID is sent in host byte order instead of network byte order. ICMP sequence generation is very similar to above, but uses a slight variation in the calculations, since it is only analyzing data from two ping packets as opposed to the six packets used in the TCP test. The generator is placed into one of six categories: all zero, all equal, random positive increments, broken increment, increments, or unclassified (ibid., ¶9). The shared sequence test is a boolean value that indicates if the host uses a unique sequence generator for TCP vs. ICMP, or if they both utilize the same counter. This value is easy to determine from the packets already classified.

The last response test we will look at is the timestamp option algorithm. nmap records the TSval and takes the difference between each consecutive TSval. Then, the difference is divided by the amount of time elapsed between the two probes which generated the responses. The result gives a rate of timestamp increments per second. All the rates are then averaged to classify the timestamp counter according

to one of four categories: unsupported option; zero timestamp; 2Hz, 100Hz, or 200Hz; or the binary logarithm of the average increase/second rounded to the nearest integer (ibid., ¶14).

To put our discussion of the sequence analysis altogether, I've included the output of `tcpdump -vv` and removed the RST packets in between connections for readability. I've also emphasized the target's IP ID field and TSval. First, the 6 TCP probes and replies:

```
21:09:30.259236 IP (tos 0x0, ttl  44, id 64959, offset 0, flags [none], proto: TCP (6),
length: 60) 10.10.2.105.44907 > 10.10.2.102.ssh: S, cksum 0x8a38 (correct), 3752614626
:3752614626(0) win 1 <wscale 10,nop,mss 1460,timestamp 4294967295 0,sackOK>

21:09:30.262516 IP (tos 0x0, ttl  64, id 17688, offset 0, flags [DF], proto: TCP (6), length:
64) 10.10.2.102.ssh > 10.10.2.105.44907: S, cksum 0x25a5 (correct), 2982307670:2982307670(0)
ack 3752614627 win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 257549798
4294967295,sackOK,eol>

21:09:30.363638 IP (tos 0x0, ttl  54, id 12468, offset 0, flags [none], proto: TCP (6),
length: 60) 10.10.2.105.44908 > 10.10.2.102.ssh: S, cksum 0x9435 (correct), 3752614627
:3752614627(0) win 63 <mss 1400,wscale 0,sackOK,timestamp 4294967295 0,eol>

21:09:30.366907 IP (tos 0x0, ttl  64, id 17689, offset 0, flags [DF], proto: TCP (6), length:
64) 10.10.2.102.ssh > 10.10.2.105.44908: S, cksum 0xdbdb (correct), 1467681637:1467681637(0)
ack 3752614628 win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 257549798
4294967295,sackOK,eol>

21:09:30.467233 IP (tos 0x0, ttl  52, id 42949, offset 0, flags [none], proto: TCP (6),
length: 60) 10.10.2.105.44909 > 10.10.2.102.ssh: S, cksum 0x9566 (correct), 3752614628
:3752614628(0) win 4 <timestamp 4294967295 0,nop,nop,wscale 5,nop,mss 640>

21:09:30.474930 IP (tos 0x0, ttl  64, id 17690, offset 0, flags [DF], proto: TCP (6), length:
60) 10.10.2.102.ssh > 10.10.2.105.44909: S, cksum 0xe57a (correct), 4028333352:4028333352(0)
ack 3752614629 win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 257549799 4294967295>

21:09:30.571334 IP (tos 0x0, ttl  40, id 30395, offset 0, flags [none], proto: TCP (6),
length: 56) 10.10.2.105.44910 > 10.10.2.102.ssh: S, cksum 0xa1ec (correct), 3752614629
:3752614629(0) win 4 <sackOK,timestamp 4294967295 0,wscale 10,eol>

21:09:30.574525 IP (tos 0x0, ttl  64, id 17691, offset 0, flags [DF], proto: TCP (6), length:
64) 10.10.2.102.ssh > 10.10.2.105.44910: S, cksum 0x8849 (correct), 114536346:114536346(0)
ack 3752614630 win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 257549799
4294967295,sackOK,eol>

21:09:30.675254 IP (tos 0x0, ttl  45, id 44651, offset 0, flags [none], proto: TCP (6),
length: 60) 10.10.2.105.44911 > 10.10.2.102.ssh: S, cksum 0x8dbe (correct), 3752614630
:3752614630(0) win 16 <mss 536,sackOK,timestamp 4294967295 0,wscale 10,eol>
```

```
21:09:30.681166 IP (tos 0x0, ttl  64, id 17692, offset 0, flags [DF], proto: TCP (6), length:
64) 10.10.2.102.ssh > 10.10.2.105.44911: S, cksum 0x8a3c (correct), 3079208175:3079208175(0)
ack 3752614631 win 65535 <mss 1460,nop,wscale 0,nop,nop,timestamp 257549799
4294967295,sackOK,eol>


21:09:30.779253 IP (tos 0x0, ttl  55, id 45331, offset 0, flags [none], proto: TCP (6),
length: 56) 10.10.2.105.44912 > 10.10.2.102.ssh: S, cksum 0xa9e2 (correct), 3752614631
:3752614631(0) win 512 <mss 265,sackOK,timestamp 4294967295 0>

21:09:30.782211 IP (tos 0x0, ttl  64, id 17693, offset 0, flags [DF], proto: TCP (6), length:
60) 10.10.2.102.ssh > 10.10.2.105.44912: S, cksum 0x1147 (correct), 3541855830:3541855830(0)
ack 3752614632 win 65535 <mss 1460,nop,nop,timestamp 257549799 4294967295,sackOK,eol>
```

And now the 2 ICMP Echos with the Echo Reply packets:

```
21:09:30.807235 IP (tos 0x0, ttl  55, id 1598, offset 0, flags [DF], proto: ICMP (1), length:
148) 10.10.2.105 > 10.10.2.102: ICMP echo request, id 51274, seq 295, length 128

21:09:30.814226 IP (tos 0x0, ttl  64, id 17694, offset 0, flags [DF], proto: ICMP (1),
length: 148) 10.10.2.102 > 10.10.2.105: ICMP echo reply, id 51274, seq 295, length 128

21:09:30.835324 IP (tos 0x4, ttl  54, id 20791, offset 0, flags [none], proto: ICMP (1),
length: 178) 10.10.2.105 > 10.10.2.102: ICMP echo request, id 51275, seq 296, length 158

21:09:30.859510 IP (tos 0x4, ttl  64, id 17695, offset 0, flags [none], proto: ICMP (1),
length: 178) 10.10.2.102 > 10.10.2.105: ICMP echo reply, id 51275, seq 296, length 158
```

We can easily see from this trace that the target uses a single integer increment for the sequence generator and that it shares the same counter between TCP and ICMP.

We have only begun to scratch the surface of all that nmap tests but due to the scope of this paper, we will stop here. However, I believe we have examined the methods and strategies used by nmap to fingerprint a device. Any reader who wishes further study of nmap would do well to read the nmap documentation which details in full all tests and possible values (*TCP/IP Fingerprinting Methods Supported by* nmap).

Xprobe2

Until I began writing this paper, I had depended solely on nmap

for all my scanning and OS detection reconnaissance. Xprobe2, however, has been a very pleasant discovery. Xprobe2 relies primarily on ICMP and was developed as a result of the "ICMP Usage in Scanning" project (Arkin, Yarochkin, 2001). Despite advertising as primarily an ICMP scanner, by default Xprobe2 will emit a small number of TCP and UDP packets during the course of testing. But these non-ICMP packets can be avoided as we will see below. However, even with this apparent contradiction, Xprobe2 is still the quietest active scanner I've witnessed. It does appear that Xprobe2 is no longer under active development, with the latest release dated July 29th, 2005. Even so, Xprobe2 is an excellent tool for fingerprinting and in my experience has often more accurately identified hosts which confused an nmap scan, especially when there are no open ports available on the target device.

Xprobe2 utilizes a "matrix based fingerprint matching based on a statistical calculation of scores for each test" (Arkin, Yarochkin, 2002, p. 7) which helps prevent false detections (or no detection) when a firewall, load balancer, or "scrubber"[1] is between the scanner and the target. Because Xprobe2 uses only valid packets (with the exception of checksums), it is very difficult to detect via host or network based IDS/IPS systems. It also performs what the authors call "fuzzy" signature matching on the results.

The logical decision tree for Xprobe2 is arranged in binary format, allowing for remarkably fast results while generating very little traffic. To start building this decision tree we will use

1 A scrubber is a piece of software that is specifically designed to alter network packets in an attempt to prevent remote fingerprinting.

Xprobe2 to generate an ICMP echo with ECN enabled and the DF bit set. Then we will examine the responses from different hosts. First, an echo request to a Windows XP SP2 box (sent from a Linux 2.6.x host):

```
14:42:36.105884 IP (tos 0x6,ECT(0), ttl  64, id 19475, offset 0, flags [DF], proto: ICMP (1),
length: 84) 192.168.0.4 > 192.168.0.101: ICMP echo request, id 19639, seq 1, length 64

14:42:36.107486 IP (tos 0x0, ttl 128, id 59791, offset 0, flags [DF], proto: ICMP (1),
length: 84) 192.168.0.101 > 192.168.0.4: ICMP echo reply, id 19639, seq 1, length 64
```

We see that the target returned the echo reply using the same flags as the request. This is the correct behavior according to RFC 792, which states "[t]o form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed." But notice the TTL value of 128, versus our initial TTL of 64. These devices are in the same broadcast domain and therefore the TTL was not decremented. Also, the target device does not support ECN, as seen by the lack of the Explicit Connection Target (ECT) option. This is not in violation of the RFC, as these values are allowed to vary per TCP stack. And now an echo request to a Linux box (sent from the same Linux host as before):

```
14:45:59.273678 IP (tos 0x6,ECT(0), ttl  64, id 49892, offset 0, flags [DF], proto: ICMP (1),
length: 84) 192.168.0.4 > 192.168.0.100: ICMP echo request, id 22065, seq 1, length 64

14:45:59.275212 IP (tos 0x6,ECT(0), ttl  64, id 56932, offset 0, flags [none], proto: ICMP
(1), length: 84) 192.168.0.100 > 192.168.0.4: ICMP echo reply, id 22065, seq 1, length 64
```

Here, the target does not follow the RFC since it did not use the same flags as the request. We also notice the reply TTL is 64 and the target supports ECN. Now, with just one probe packet we can eliminate entire classes of devices, solely based on these three pieces of information.

To complete the decision tree, this process is repeated through the various ICMP query, response, and error types and the values

recorded.

Xprobe2 is designed in a modular fashion, which makes it very flexible in its operation, even allowing for the custom development of modules (*modules_howto.txt*, 2005). Any module can be disabled at runtime and a listing of available modules can be displayed with the `-L` option. Xprobe2 will not perform a port scan by default, but will do so if the `-T` or `-U` options list the ports to be scanned. In addition, the `-B` option will force an attempt at a TCP handshake to find an open or closed port. Known port states can be given with the `-p` option, like `-p tcp:80:open`.

The first step of a default scan is to make sure the target box is up, via an ICMP Echo request and then a TCP SYN packet. Both of these actions are considered a module in Xprobe2 and can be disabled just like any other module. However, at least one of these must be enabled for Xprobe2 to run.

Once it is determined that the target host is up, the remainder of the enabled modules will run. Each module operates independently from the others. In other words, the test results of one module don't affect the results of any other module. In this way, the "fuzziness" of the matching algorithm is ensured. The results are scored in a matrix-like table which resembles:

	OS 1	**OS 2**	**OS 3**	**OS *i***
Test 1 (TTL)	score	score	score	...
Test 2 (IP_ID)	score	score	score	...
Test 3 (ICMP Port unreachable)	score	score	score	...
Test n	...	...	...	...
Totals	X	Y	Z	D

Each test is conducted independently and the possible values are YES(3), PROBABLY_YES(2), PROBABLY_NO(1), and NO(0). "This approach gives us probabilistic support since the highest score given for an OS (or OSs) is the most likely to produce an accurate match" (Arkin & Yarochkin, 2002, p7).

When the scan is complete the scores for each test are compared to the signature database and a probable OS is determined by the database entry which most closely resembles the results seen on the wire. Xprobe2, by default, will print other OS's that also might match and a guess probability associated with each listing.

A sample run of Xprobe2 using all the defaults against a Mac running 10.4.10 follows[1]:

```
jm@linuxbox> sudo xprobe2 10.10.2.100

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

[+] Target is 10.10.2.100
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [10] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [11] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[x] [12] fingerprint:smb  -  SMB fingerprinting module
[x] [13] fingerprint:snmp  -  SNMPv2c fingerprinting module
[+] 13 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.10.2.100. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.10.2.100. Module test failed
[-] No distance calculation. 10.10.2.100 appears to be dead or no ports known
[+] Host: 10.10.2.100 is up (Guess probability: 50%)
[+] Target: 10.10.2.100 is alive. Round-Trip Time: 0.00396 sec
```

1 You will note that Xprobe2 also requires root privileges

```
[+] Selected safe Round-Trip Time value is: 0.00792 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
[+] Primary guess:
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.4.1" (Guess probability: 100%)
[+] Other guesses:
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.4.0" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.9" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.8" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.7" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.6" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.5" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.4" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.0" (Guess probability: 100%)
[+] Host 10.10.2.100 Running OS: "Apple Mac OS X 10.3.1" (Guess probability: 100%)
[+] Cleaning up scan engine
[+] Modules deinitialized
[+] Execution completed.
```

As you can see, Xprobe2 is fairly verbose by default, and that's without specifying the verbose option! You can also see the repercussions of the lack of signature updates, since the oldest version listed is 10.4.1, which a search of the fingerprint database reveals to be the latest version included.

However, in cases where stealth operation is important, an ICMP only scan from Xprobe2 can still be amazingly accurate. This is achieved by only enabling the ping:icmp_echo, infogather:ttl_calc, and fingerprint:icmp_echo modules (module numbers 1, 4, and 6 respectively). Below are nmap and Xprobe2 scans to the same target:

```
jm@linuxbox> sudo nmap -sV -P0 -O 10.92.203.179

Starting Nmap 4.20 ( http://insecure.org ) at 2007-08-27 16:18 CDT
Warning:  OS detection for 10.92.203.179 will be MUCH less reliable because we did not find
at least 1 open and 1 closed TCP port
All 1697 scanned ports on 10.92.203.179 are filtered
MAC Address: 00:03:93:02:17:7E (Apple Computer)
Device type: general purpose
Running: Apple Mac OS X 10.3.X|10.4.X|10.5.X, FreeBSD 4.x
OS details: Applie Mac OS X 10.3.9 - 10.4.7, Apple Mac OS X 10.4.8 (Tiger), Apple Mac OS X
10.4.8 (Tiger) (PPC), OS X Server 10.5 (Leopard) pre-release build 9A284, FreeBSD 4.10-
RELEASE (x86)

jm@linuxbox> sudo xprobe2 -M 1 -M 4 -M 6 10.92.203.179

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

```
[+] Target is 10.92.203.179
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [3] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[+] 3 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] No distance calculation. 10.92.203.179 appears to be dead or no ports known
[+] Host: 10.92.203.179 is up (Guess probability: 50%)
[+] Target: 10.92.203.179 is alive. Round-Trip Time: 0.00106 sec
[+] Selected safe Round-Trip Time value is: 0.00212 sec
[+] Primary guess:
[+] Host 10.92.203.179 Running OS: "Apple Mac OS X 10.3.8" (Guess probability: 100%)
```

Note that both scanners returned the same class of device, but Xprobe2 did it by sending only 2 packets!

```
00:05:17.948438 IP (tos 0x0, ttl  64, id 65286, offset 0, flags [none], proto: ICMP (1),
length: 84) 10.10.2.101 > 10.10.2.100: ICMP echo request, id 65286, seq 0, length 64

00:05:17.949945 IP (tos 0x0, ttl  64, id 17307, offset 0, flags [none], proto: ICMP (1),
length: 84) 10.10.2.100 > 10.10.2.101: ICMP echo reply, id 65286, seq 0, length 64

00:05:17.984297 IP (tos 0x6,ECT(0), ttl  64, id 51387, offset 0, flags [DF], proto: ICMP (1),
length: 84) 10.10.2.101 > 10.10.2.100: ICMP echo request, id 65286, seq 1, length 64

00:05:17.985859 IP (tos 0x6,ECT(0), ttl  64, id 17308, offset 0, flags [DF], proto: ICMP (1),
length: 84) 10.10.2.100 > 10.10.2.101: ICMP echo reply, id 65286, seq 1, length 64
```

There does appear to be some quirks in Xprobe2's processing, however. Looking at our first scan of a Mac target host, I noticed the smb and snmp scans printed out an ugly warning

```
[-] fingerprint:smb need either TCP port 139 or 445 to run
[-] fingerprint:snmp: need UDP port 161 open
```

Xprobe2 did not run the modules because I had not specified in the command that the required ports were open[1]. This is verified by the network capture:

```
12:14:46.509586 IP 10.10.2.101 > 10.10.2.100: ICMP echo request, id 5298, seq 0, length 64
12:14:46.512572 IP 10.10.2.100 > 10.10.2.101: ICMP echo reply, id 5298, seq 0, length 64
12:14:46.549146 IP 10.10.2.101 > 10.10.2.100: ICMP echo request, id 5298, seq 1, length 64
12:14:46.550720 IP 10.10.2.100 > 10.10.2.101: ICMP echo reply, id 5298, seq 1, length 64
12:14:46.572612 IP 10.10.2.101 > 10.10.2.100: ICMP time stamp query id 5298 seq 0, length 20
12:14:47.024413 IP 10.10.2.101 > 10.10.2.100: ICMP address mask request, length 12
```

1 One could argue that because I enabled the modules requesting SMB and SNMP tests, Xprobe2 should probe the ports for me, but that's another matter.

```
12:14:48.084506 IP 10.10.2.101.domain > 10.10.2.100.65534:  21180$ 1/0/0 (76)
12:14:48.086172 IP 10.10.2.100 > 10.10.2.101: ICMP 10.10.2.100 udp port 65534 unreachable,
length 36
12:14:48.115719 IP 10.10.2.101.60214 > 10.10.2.100.65535: S 957525460:957525460(0) win 6840
12:14:48.117240 IP 10.10.2.100.65535 > 10.10.2.101.60214: R 0:0(0) ack 957525461 win 0
12:14:48.117293 IP 10.10.2.101.63698 > 10.10.2.100.65535: S 1316841687:1316841687(0) win 6840
12:14:48.118623 IP 10.10.2.100.65535 > 10.10.2.101.63698: R 0:0(0) ack 1316841688 win 0
```

We see a spoofed DNS reply packet which illicits an ICMP unreachable packet. And we also see two TCP SYN packets to random high ports. But there are no packets to SNMP or SMB ports (UDP port 161 and TCP ports 139 or 445 respectively).

So I decided to remove the modules for those tests, using the `-D` option, since I already knew neither SMB or SNMP were running on that box. The result was a little surprising:

```
jm@linuxbox> sudo xprobe2 -D13 -D14 10.10.2.100

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

[+] Target is 10.10.2.100
[+] Loading modules.
[+] Following modules are loaded:
[x] [1] ping:icmp_ping  -  ICMP echo discovery module
[x] [2] ping:tcp_ping  -  TCP-based ping discovery module
[x] [3] ping:udp_ping  -  UDP-based ping discovery module
[x] [4] infogather:ttl_calc  -  TCP and UDP based TTL distance calculation
[x] [5] infogather:portscan  -  TCP and UDP PortScanner
[x] [6] fingerprint:icmp_echo  -  ICMP Echo request fingerprinting module
[x] [7] fingerprint:icmp_tstamp  -  ICMP Timestamp request fingerprinting module
[x] [8] fingerprint:icmp_amask  -  ICMP Address mask request fingerprinting module
[x] [9] fingerprint:icmp_info  -  ICMP Information request fingerprinting module
[x] [10] fingerprint:icmp_port_unreach  -  ICMP port unreachable fingerprinting module
[x] [11] fingerprint:tcp_hshake  -  TCP Handshake fingerprinting module
[x] [12] fingerprint:tcp_rst  -  TCP RST fingerprinting module
[+] 12 modules registered
[+] Initializing scan engine
[+] Running scan engine
[-] ping:tcp_ping module: no closed/open TCP ports known on 10.10.2.100. Module test failed
[-] ping:udp_ping module: no closed/open UDP ports known on 10.10.2.100. Module test failed
[-] No distance calculation. 10.10.2.100 appears to be dead or no ports known
[+] Host: 10.10.2.100 is up (Guess probability: 50%)
[+] Target: 10.10.2.100 is alive. Round-Trip Time: 0.00212 sec
[+] Selected safe Round-Trip Time value is: 0.00423 sec
[-] fingerprint:tcp_hshake Module execution aborted (no open TCP ports known)
[+] Primary guess:
[+] Host 10.10.2.100 Running OS: "FreeBSD 5.2" (Guess probability: 96%)
```

The packets sent across the wire for the second scan were identical to the first with one addition after the ICMP address mask

request:

```
14:03:40.020736 IP 10.10.2.101 > 10.10.2.100: ICMP information request, length 8
```

It seems that the ICMP information module isn't loaded if the SMB or SNMP modules are enabled. But if these two modules are disabled the additional ICMP packet is generated, the result of which causes Xprobe2 to conclude the target is FreeBSD instead of Mac OS X.

In my personal tests, I have become accustomed executing the ICMP only modules I listed previously. Using these modules reduces the network footprint to the absolute minimum and still seems to yield a remarkably high accuracy rate.

p0f – A Passive Scanner

p0f is a passive OS detection engine, which reads packets from the network and analyzes them without generating any traffic of its own. If stealth operation is important in a particular scenario, passive fingerprinting is your best option. p0f was originally based on Siphon, an early implementation of passive scanning (Zalewski, 2006).

p0f can operate in one of four different modes: SYN, SYN+ACK, RST, and stray ACK (Zalewski, 2006, Section 1 ¶4). It should be noted that each method also has a different fingerprint database and not all the databases are considered equal.

The default is to listen for incoming connections and only fingerprint those clients that are making a connection to the host running p0f, hence the name SYN mode. This is definitely the most comprehensive and accurate database of the four modes, and the only

mode we will seriously examine. SYN+ACK mode will perform outgoing connection fingerprinting and can be used when you want to know what sort of server you (or your users) are connecting to. RST mode can be used when it's not possible to establish a full connection to the target device. In this case, a SYN packet is sent to a closed port and the resulting RST packet is analyzed. Lastly, stray ACK can be used to fingerprint an existing connection, and is described as "absolutely experimental" (Zalewski, 2006, Section 4, description of options).

p0f uses a surprisingly large number of signature traits to fingerprint any given system. Among the options we've already discussed are TTL, DF bit, timestamps, and window scaling. p0f stores the SYN mode fingerprint database in a file called *p0f.fp*, which by default is kept in the */etc/p0f* directory. The fingerprint file contains a very thorough explanation of all TCP traits that are examined. According to this file, in addition to the items above, p0f measures the overall packet size, selective ACK (SACK) support, NOP option, EOL option, the sequence of TCP options, other options p0f doesn't recognize, and various "quirks" (as they are referred to by p0f) in the TCP stack. p0f also examines the Maximum Segment Size (MSS) field to determine the link type of the originating packet.

As an example, I ran tcpdump on a web server for a short while and captured only packets with the SYN flag set, which restricts the packets captured to the first two steps of the TCP handshake. I then ran p0f against the generated libpcap file. The `-s` option instructs p0f to read from a file instead of listening on a live interface. Note that IP addresses have been munged and, to conserve space, the

output of hosts has been limited to a few that displayed interesting results.

```
jm@linuxbox> p0f -s p0f_samples.pcap
p0f - passive os fingerprinting utility, version 2.0.5
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'p0f_samples.pcap', 231 sigs (13 generic), rule: 'all'.
69.5.x.y:35087 - UNKNOWN [64512:112:1:48:M1360,N,N,N,N:.:?:?]
  -> 10.1.1.1:80 (link: (Google/AOL))
75.88.x.y:60381 - Windows XP/2000 (RFC1323, w+, no tstamp) [GENERIC]
  Signature: [65535:119:1:52:M1380,N,W3,N,N,S:.:Windows:?]
  -> 10.1.1.1:80 (distance 9, link: GPRS, T1, FreeS/WAN)
10.41.x.y:3652 - Windows XP SP1, 2000 SP3 (2)
  -> 10.1.1.1:80 (distance 4, link: ethernet/modem)
128.194.x.y:1570 - Windows 2000 SP2+, XP SP1 (seldom 98 4.10.2222)
  -> 10.1.1.1:80 (distance 19, link: PIX, SMC, sometimes wireless)
205.123.x.y:1780 - Windows 2000 SP2+, XP SP1 (seldom 98 4.10.2222)
  -> 10.1.1.1:80 (distance 2, link: ISDN ppp)
66.249.x.y:33556 - UNKNOWN [5720:55:1:60:M1380,S,T,N,W6:.:?:?] [Tiscali Denmark] (up: 10110
hrs)
  -> 10.1.1.1:80 (link: GPRS, T1, FreeS/WAN)
71.241.x.y:3393 - Windows 2000 SP4, XP SP1 (firewall!)
  -> 10.1.1.1:80 (distance 9, link: PIX, SMC, sometimes wireless)
64.136.x.y:21211 - OpenBSD 3.0-3.4 (up: 4088 hrs)
  -> 10.1.1.1:80 (distance 9, link: GPRS, T1, FreeS/WAN)
122.152.x.y:50224 - Linux 2.5 (sometimes 2.4) (4) (NAT!) (up: 2981 hrs)
  -> 10.1.1.1:80 (distance 10, link: GPRS, T1, FreeS/WAN)
```

Even for hosts that are listed as “UNKNOWN”, p0f is able to determine network distance, link type, the probable presence of a NAT device between the remote and local hosts, and sometimes an uptime estimate. The p0f README file states the NAT determination relies on the detection of tweaked MSS values, but as with other metrics, this may not always be accurate. “Most likely, the reason for [an adjusted MSS] is indeed a NATing router, but there are some other explanations. Linux, for example, tends to mix up MTUs from different interfaces in certain scenarios” (Zalewski, 2006, Section 8, ¶4).

p0f can also be run as a daemon. This can be very useful for network based OS detection. Consider the scenario as depicted in Figure 4.1. Here, the firewall sends a copy of the packets to be analyzed to the device running p0f in daemon mode. In this way,

every host communicating through the firewall can be examined without

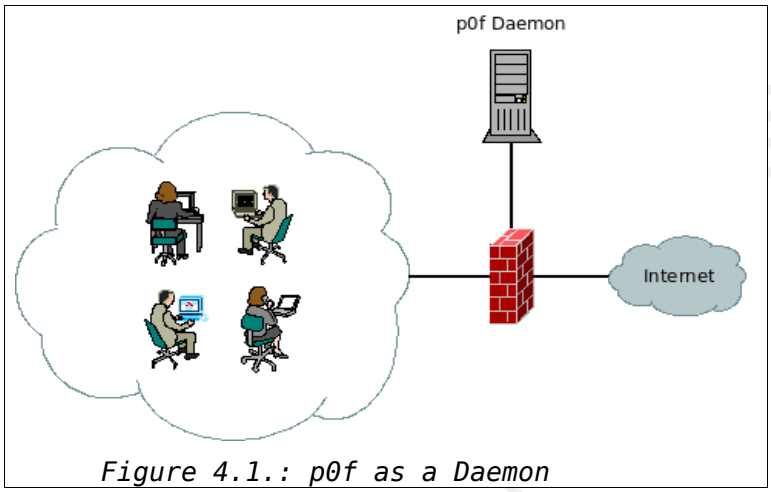

Figure 4.1.: p0f as a Daemon

any undue strain on critical inline network equipment. Or you could remove the firewall from the diagram and put in its place a core router/switch, or even a network appliance. Now instead of sending *every* packet to the p0f daemon, the appliance could send only the packets from those devices it wishes to fingerprint.

SYN+ACK mode has a significantly reduced fingerprint database, as seen by the following run:

```
jm@linuxbox> sudo p0f -i eth1 -A
p0f - passive os fingerprinting utility, version 2.0.5
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN+ACK) on 'eth1', 57 sigs (1 generic), rule: 'all'.
68.178.x.y:80 - Linux recent 2.4 (1) (up: 3255 hrs)
  -> 192.168.1.68:35020 (distance 16, link: GPRS, T1, FreeS/WAN)
68.178.x.y:80 - UNKNOWN [S4:49:1:60:M1452,S,T,N,W5:A:?:?] (NAT!) (up: 1 hrs)
  -> 192.168.1.68:41500 (link: pppoe (DSL))
216.239.x.y:80 - UNKNOWN [8190:244:0:44:M1452:A:?:?] [Cable.BG / Teleca.SE]
  -> 192.168.1.68:36709 (link: pppoe (DSL))
72.14.x.y:80 - UNKNOWN [5672:50:0:60:M1430,S,T,N,W6:AT:?:?] [Tiscali Denmark] (up: 10766 hrs)
  -> 192.168.1.68:50176 (link: unknown-1470)
209.85.x.y:80 - UNKNOWN [5672:49:0:60:M1430,S,T,N,W6:AT:?:?] [tos 224] (up: 10916 hrs)
  -> 192.168.1.68:57138 (link: unknown-1470)
204.245.x.y:80 - Linux recent 2.4 (1) (up: 852 hrs)
  -> 192.168.1.68:37640 (distance 12, link: ethernet/modem)
72.3.x.y:443 - UNKNOWN [65535:50:1:64:M1380,N,W1,N,N,T,S,E:PAT:?:?] (up: 4296 hrs)
  -> 192.168.1.68:37299 (link: GPRS, T1, FreeS/WAN)
67.137.x.y:80 - Linux recent 2.4 (1) (up: 8602 hrs)
  -> 192.168.1.68:53388 (distance 11, link: ethernet/modem)
128.241.x.y:80 - UNKNOWN [S3:242:0:44:M1280:A:?:?]
  -> 192.168.1.68:50310 (link: unknown-1320)
```

A much greater percentage of hosts are classified as 'UNKNOWN', but there is still a decent amount of information about each of these devices. We also see some of the masquerade detection in action, as the first two hosts had the same address, but different fingerprints.

We can see that passive fingerprinting can be an extremely valuable practice, particularly for network administrators. Passive fingerprinting also requires very little resources while yielding a large amount of information.

5. How Do I Prevent Successful Fingerprints?

Preventing OS fingerprinting is only necessary in those cases where malicious reconnaissance is a concern. This is almost always the case on the edge of the network. But also consider the situation where someone is already on your network and wishes not to be found, he will try and blend in with the rest of the devices on your network. In that case, you should be aware of the possible options and behaviors to watch for.

Defeating OS detection is a daunting task, as you might have already surmised. Given the several methods of detection, one might think it nearly impossible to block all fingerprinting, and you may be right. I believe this case reflects the rest of the security industry, in that perfection may not be possible, but we can at least make it slightly harder to exploit our weaknesses. There are not a lot of options in this area and some administrators might view the steps outlined below and determine the effort required to implement the remedy is too great in relation to the risk. Some of these steps I hope are simply review and might seem obvious, nonetheless, they

are included here for completeness.

The first step is to make sure that external hosts are not able to directly scan internal targets. ICMP should only be allowed if your firewall maintains stateful connections for ICMP in the same way almost all firewalls do for TCP. Even then, only required ICMP types should be allowed and only in specific directions and to/from required hosts. Note that even the best configurations will probably not prevent passive fingerprinting, especially of network servers that are designed to accept connections from the Internet.

Host level modifications are possible, but do not typically scale well. TCP/IP parameters can be modified to make the device look like another OS. This will cause "script kiddie" type scans to be fooled, but will not deter a skilled attacker. And obviously, if you are using OS scanners to inventory your network, you would not want to employ these measures on the internal devices anyway. **One other word of warning:** changing TCP/IP settings will not only change how the network traffic appears, they **will** have an impact on your device's network performance. Sometimes that impact could be good, but sometimes it could be bad. Always keep this fact in mind and watch for the effects when you are adjusting the settings below.

In Windows, these settings are found in the registry, under `HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`. Settings and values to examine might include: `TcpWindowSize`, `Tcp1323Opts`, `DefaultTTL`, `EnablePMTUDiscovery`, and `MTU` (*Windows 2000/XP Registry Tweaks*, 2001).

There are third-party programs that are specifically designed to

make these modifications easier. One such pair of programs are “TCP Optimizer” and “Windows 2000/XP registry patches” from SpeedGuide.net (ibid). Another product is ServerMask from Port80Software. ServerMask also performs application masking, which include tasks like removing (or changing to an arbitrary value) the IIS or Exchange headers from the Microsoft's web or mail servers, respectively (*ServerMask*).

On linux, TCP/IP settings are set via the `/proc` filesystem, under the `/proc/sys/net/ipv4`, and `/proc/sys/net/core` directories. Values in the `/proc/sys` directory can be controlled by `sysctl` and `sysctl.conf`. Items that might be of interest include `icmp_default_ttl`, `tcp_ecn`, `tcp_sack`, `tcp_timestamps`, and `tcp_window_scaling`.

Banner grabbing should be a bit easier to defend against. The Apache config file allows you to limit the information listed in the header. Or the mod_headers Apache module can be used to spoof another web server completely (*mod_headers*). Most SMTP mailers allow you to suppress software and version info, though note that according to RFC 2821, Section 4.1.1.1, the server host name and the strings 'SMTP' or 'ESMTP' are required in the server banner.

Almost surprisingly, SSH servers seem to be the exception in configuration for this situation. It seems, however, that the version information is required for proper client/server negotiation to work around incompatibilities between protocol versions (*OpenSSH FAQ*, Section 2.14). If, however, you still wish to remove the version (or in some cases the distribution the server is running),

you will be required to download the source code and edit the `version.h` file and recompile. Alternately, Henrik Kramshøj has provided a patch to apply to the source code that provides the option `HideVersion` in `sshd_config` (Kramshøj, 2007). This patch changes the version string to `SSH-2.0-HIDDEN`. Both approaches require recompiling from source every time an upgrade is needed.

The best advice for any further application hardening is to perform scans against your own network (always with permission!) and document your findings. Then begin the search in the relevant software documentation for the required changes. Always be sure you understand the implications of any changes you plan on implementing before moving forward.

6. Conclusions and Final Comments

There are several relatively simple methods of detecting a target's operating system. And there are many excellent, free network scanners for a network administrator to employ when mapping and maintaining a network. But no scanner is perfect. I usually find myself running nmap and then running Xprobe2 – or running Xprobe2 and then running nmap! p0f is excellent for maintaining a solid inventory of devices that communicate through an edge device – or through a central route. Each scanner is created with specific purposes in mind and therefore have unique strengths and weaknesses. A wise network administrator will learn how to use several scanners – and learn the good and bad points of each - and practice with them until she is comfortable using all of them and knows which situation requires a specific scanner.

Because these scanners are so readily available, it should be obvious that white-hat administrators will not be the only individuals using them on your network. You should know these scanners are launched against your networks, and probably on a regular basis. And because the most basic of traffic reveals so much about the devices on the network, firewall and edge router ACLs should be maintained to allow only that traffic which is absolutely essential to production – even ICMP should be carefully restricted.

7. References

Arkin, O. (2001, May 5). Fun with IP Identification Field Values (Identifying Older MS Based OSs). Retrieved August 25, 2007, from Bugtraq Mailing List Archives Web site: http://seclists.org/bugtraq/2001/May/0044.html

Arkin, O. (2001, May 15). Fingerprinting linux kernel 2.4.x based machines using ICMP (and IPID). Retrieved August 25, 2007, from Bugtraq Mailing List Archives Web site: http://seclists.org/bugtraq/2001/May/0135.html

Arkin, O. (2001, June). ICMP usage in scanning. Retrieved August 26, 2007, from The Sys-Security Group Web site: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf

Arkin, O, & Yarochkin, F (2001, August 12). ICMP based remote OS TCP/IP stack fingerprinting techniques. *Phrack #57*, Retrieved July 21, 2007, from Phrack Magazine Web site: http://phrack.org/archives/57/p57-0x07.

Arkin, O., & Yarochkin, F. (2001). X remote ICMP based OS fingerprinting techniques. Retrieved April 19, 2007, from The Sys-Security Group Web site: http://www.sys-security.com/archive/papers/X_v1.0.pdf.

Arkin, O & Yarochkin, F (2002). Xprobe v2.0 A "Fuzzy" Approach to Remote Active Operating System Fingerprinting. Retrieved April 19, 2007 from Web site: http://www.sys-security.com/archive/papers/Xprobe2.pdf

Biancuzzi, F (2006, Jan 31). Nmap 4.00 with Fyodor. *Security Focus*, Retrieved July 21, 2007, from Security Focus Web site: http://www.securityfocus.com/columnists/384

Fingerprint (n.d.). Retrieved April 28, 2007, from Wikipedia Web site: http://en.wikipedia.org/wiki/Fingerprint

Fyodor (1997, Sep 1). The Art of Port Scanning. Retrieved August 25, 2007, from Phrack Magazine Web site: http://www.phrack.org/issues.html?issue=51&id=11&mode=txt

German, E. (n.d.). Fingerprint FAQ. Retrieved April 28, 2007, from Latent Print Examination Web site: http://onin.com/fp/lpfaq.html#q1jg

ISS Security Advisory (1998, Dec 10). Retrieved September 9, 2007, from Internet Security Systems Web site: http://xforce.iss.net/xforce/alerts/id/advise15

Kramshøj, H (2007, January 22). OpenSSH HideVersion patch. Retrieved September 8, 2007, from Web site:

http://www.kramse.dk/projects/unix/opensshhideversion_en.html

Kozierok, C (2005, September 20). TCP connection establishment sequence number synchronization and parameter exchange. Retrieved August 4, 2007, from The TCP/IP Guide Web site: http://www.tcpipguide.com/free/t_TCPConnectionEstablishmentSequenceNumberSynchroniz.htm

Lasser, J (2002, January 30). Passive Aggressive. Retrieved September 1, 2007, from SecurityFocus Web site: http://www.securityfocus.com/columnists/57

modules_howto.txt (2005, June 26). Xprobe2 source code. Retrieved May 26, 2007 from SourceForge Web site: http://prdownloads.sourceforge.net/xprobe/xprobe2-0.3.tar.gz

mod_headers (n.d.). Retrieved September 8, 2007 from Apache HTTP Server Web site: http://httpd.apache.org/docs/2.2/mod/mod_headers.html

nmap - Free Security Scanner For Network Exploration & Security Audits (n.d.). Retrieved April 14, 2007, from Web site: http://insecure.org/nmap/index.html

OpenSSH FAQ (2005, September 20). Retrieved September 8, 2007, from OpenSSH Web site: http://www.openssh.org/faq.html

Perens, B (1998). Why Security-Through-Obscurity Won't Work. Retrieved May 26, 2007, from Slashdot Web site: http://slashdot.org/features/980720/0819202.shtml

RFC 791 (1981, Sept). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc791.html

RFC 792 (1981, Sept). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc792.html

RFC 793 (1981, Sept). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc793.html

RFC 1122 (1989, Oct). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc1122.html

RFC 1323 (1992, May). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc1323.html

RFC 1349 (1992, July). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc1349.html

RFC 1948 (1996, May). Retrieved August 4, 2007, from Web site: http://www.faqs.org/rfcs/rfc1948.html

RFC 2821 (2001, April). Retrieved September 8, 2007, from Web site: http://www.faqs.org/rfcs/rfc1948.html

RFC 3168 (2001, Sept). Retrieved September 9, 2007, from Web site: http://www.faqs.org/rfcs/rfc3168.html

ServerMask (n.d.). Retrieved September 13, 2007, from Port80Software Web site: http://www.port80software.com/products/servermask/

TCP/IP Fingerprinting Methods Supported by nmap (n.d.). Retrieved April 14, 2007, from Insecure.Org Web site:

http://insecure.org/nmap/osdetect/osdetect-methods.html

US-CERT (2007, Jan 24). Vulnerability Note VU#341288. Retrieved April 28, 2007, from US-CERT Web site: http://www.kb.cert.org/vuls/id/341288

Windows 2000/XP Registry Tweaks (2001, March 31). Retrieved September 8, 2007, from SpeedGuide.net Web site: http://www.speedguide.net/read_articles.php?id=157

Zalewski, M (2001, April). Strange attractors and TCP/IP sequence number analysis. Retrieved August 4, 2007, from Web site: http://lcamtuf.coredump.cx/oldtcp/tcpseq.html

Zalewski, M (2006, Sept 6). The new p0f. Retrieved July 21, 2007, from Web site: http://lcamtuf.coredump.cx/p0f.shtml

Zalewski, M (2006). p0f README Retrieved May 5, 2007 from Web site: http://lcamtuf.coredump.cx/p0f/README

Appendix A: Header Diagrams

Here for reference, I've included diagrams of the protocol headers, with the gracious permission of Matt Baxter. The full, printable, diagrams can be downloaded at http://www.fatpipe.org/~mjb/Drawings

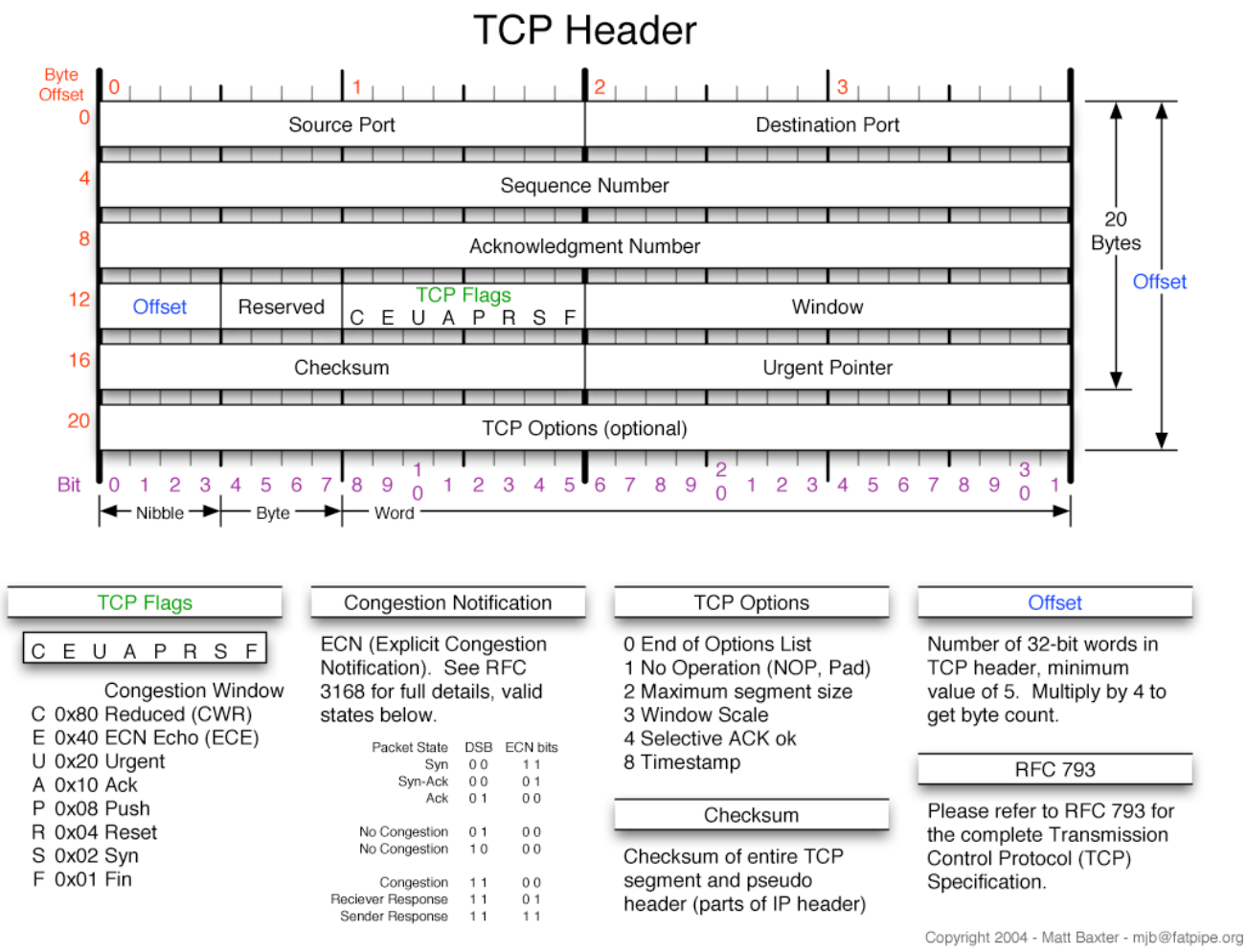

Figure 7.1: TCP Header

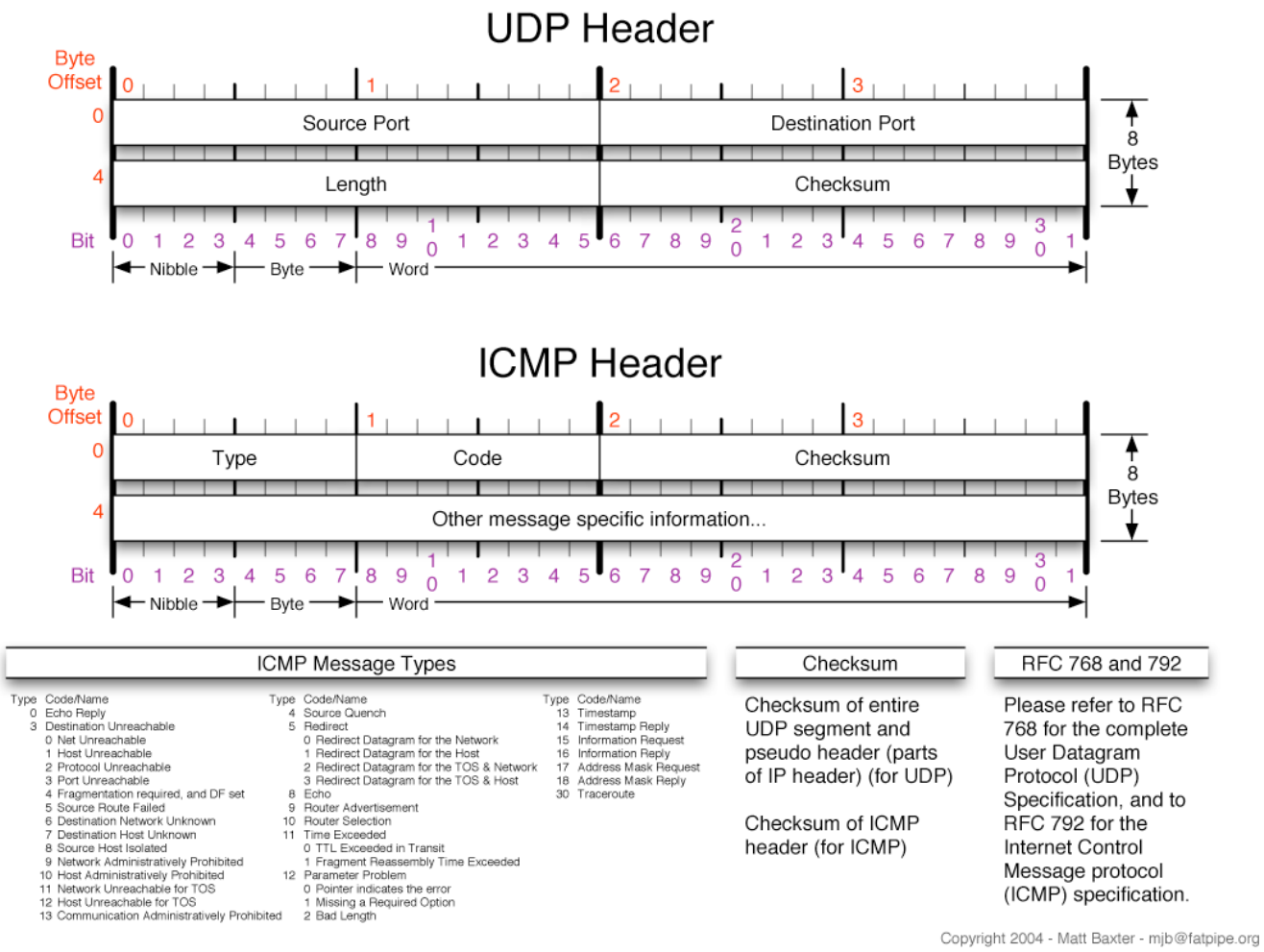

Figure 7.2: UDP and ICMP Headers

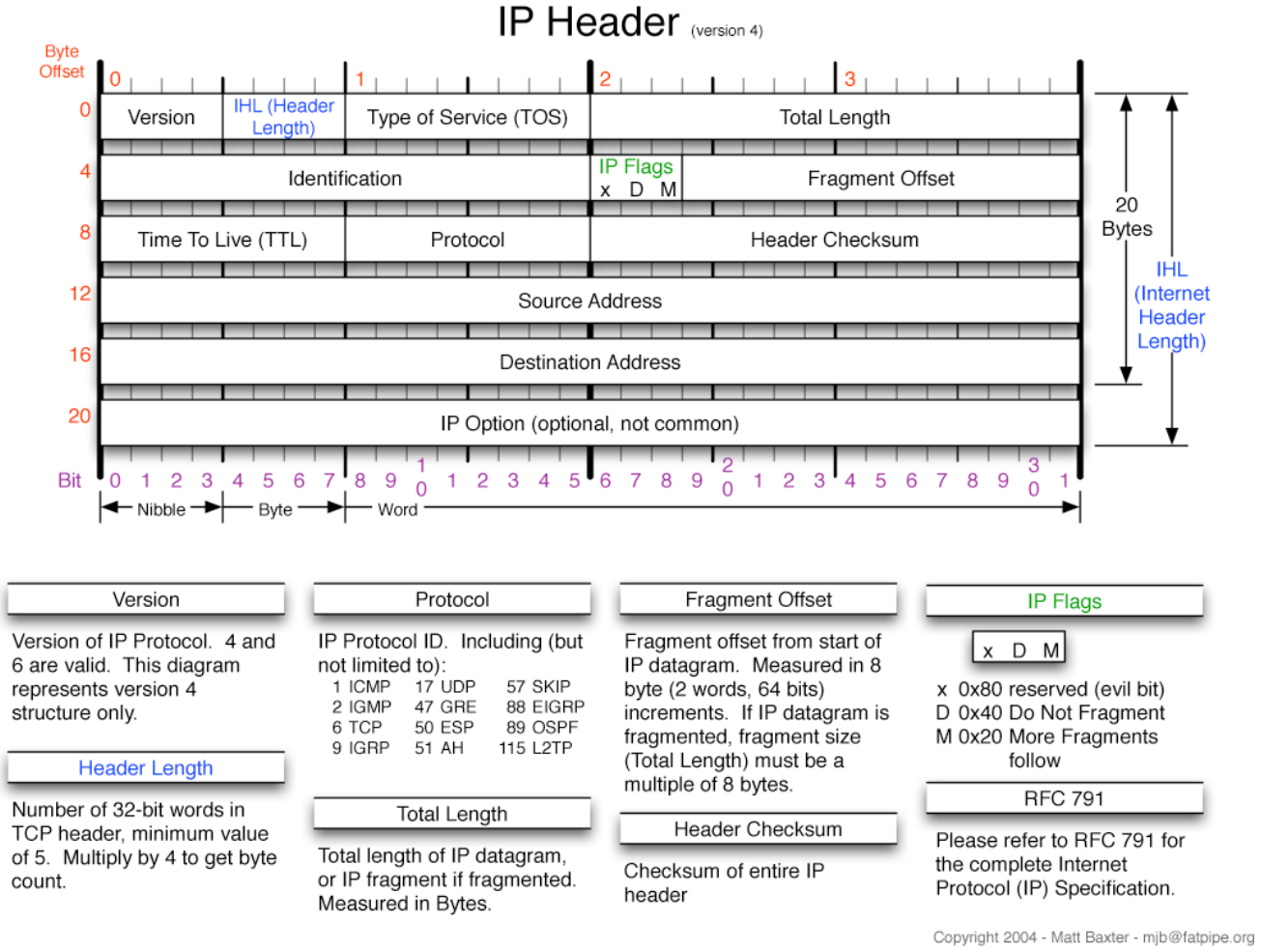

Figure 7.3: IP Header

Last Updated: December 16th, 2018

Upcoming SANS App Sec Training

Community SANS Denver DEV540	Denver, CO	Jan 14, 2019 - Jan 18, 2019	Community SANS
SANS Las Vegas 2019	Las Vegas, NV	Jan 28, 2019 - Feb 02, 2019	Live Event
SANS Security East 2019	New Orleans, LA	Feb 02, 2019 - Feb 09, 2019	Live Event
SANS Anaheim 2019	Anaheim, CA	Feb 11, 2019 - Feb 16, 2019	Live Event
SANS Zurich February 2019	Zurich, Switzerland	Feb 18, 2019 - Feb 23, 2019	Live Event
SANS Riyadh February 2019	Riyadh, Kingdom Of Saudi Arabia	Feb 23, 2019 - Feb 28, 2019	Live Event
Community SANS Nashville DEV541	Nashville, TN	Feb 26, 2019 - Mar 01, 2019	Community SANS
SANS San Francisco Spring 2019	San Francisco, CA	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS London March 2019	London, United Kingdom	Mar 11, 2019 - Mar 16, 2019	Live Event
SANS Munich March 2019	Munich, Germany	Mar 18, 2019 - Mar 23, 2019	Live Event
Community SANS Chantilly DEV541	Chantilly, VA	Mar 25, 2019 - Mar 28, 2019	Community SANS
SANS 2019	Orlando, FL	Apr 01, 2019 - Apr 08, 2019	Live Event
Cloud Security Summit & Training 2019	San Jose, CA	Apr 29, 2019 - May 06, 2019	Live Event
Security West 2019 - DEV522: Defending Web Applications Security Essentials	San Diego, CA	May 09, 2019 - May 14, 2019	vLive
SANS Security West 2019	San Diego, CA	May 09, 2019 - May 16, 2019	Live Event
Community SANS Austin DEV540	Austin, TX	May 20, 2019 - May 24, 2019	Community SANS
Community SANS Vancouver DEV540	Vancouver, BC	Jun 10, 2019 - Jun 14, 2019	Community SANS
SANSFIRE 2019	Washington, DC	Jun 15, 2019 - Jun 22, 2019	Live Event
SANSFIRE 2019 - DEV540: Secure DevOps and Cloud Application Security	Washington, DC	Jun 17, 2019 - Jun 21, 2019	vLive
SANS Cyber Defence Canberra 2019	Canberra, Australia	Jun 24, 2019 - Jul 13, 2019	Live Event
SANS San Francisco Summer 2019	San Francisco, CA	Jul 22, 2019 - Jul 27, 2019	Live Event
SANS Boston Summer 2019	Boston, MA	Jul 29, 2019 - Aug 03, 2019	Live Event
SANS San Jose 2019	San Jose, CA	Aug 12, 2019 - Aug 17, 2019	Live Event
SANS Brussels September 2019	Brussels, Belgium	Sep 02, 2019 - Sep 07, 2019	Live Event
SANS Munich September 2019	Munich, Germany	Sep 02, 2019 - Sep 07, 2019	Live Event
SANS Paris September 2019	Paris, France	Sep 16, 2019 - Sep 21, 2019	Live Event
SANS London September 2019	London, United Kingdom	Sep 23, 2019 - Sep 28, 2019	Live Event
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced