

1.1 PURPOSE

The purpose of this paper is to explain the details and techniques that can be used in passive OS fingerprinting. In this paper, we will look at packets captured by TCPDUMP. Although this paper assumes the reader has a basic understanding of the Transmission Control Protocol/Internet Protocol (TCP/IP), we will briefly cover certain fields in the TCP/IP header. This paper will specifically cover passive OS fingerprinting using TCP and the SYN packet. We will cover other flags but the majority of the work has been done analyzing a SYN packet. Passive OS fingerprinting using ICMP has been covered in great detail at <http://www.sys-security.com/>.

Before we begin, I want to clear up any confusion between passive Operating System (OS) fingerprinting (what we are doing here) and active OS fingerprinting (tools like NMAP use this technique). First, passive OS fingerprinting does not send out any packets. All that is needed to perform passive OS fingerprinting is a computer, a sniffer and a desire to dig into a packet. Active OS fingerprinting on the other hand will send special or not so special packets to a machine and wait for a response from that machine. That response will determine what operating system the remote system is running.

1.2 HEADER OVERVIEW

Before we cover passive OS fingerprinting, let us look at the TCP and IP headers. Figure 1 shows an example of an IP header.

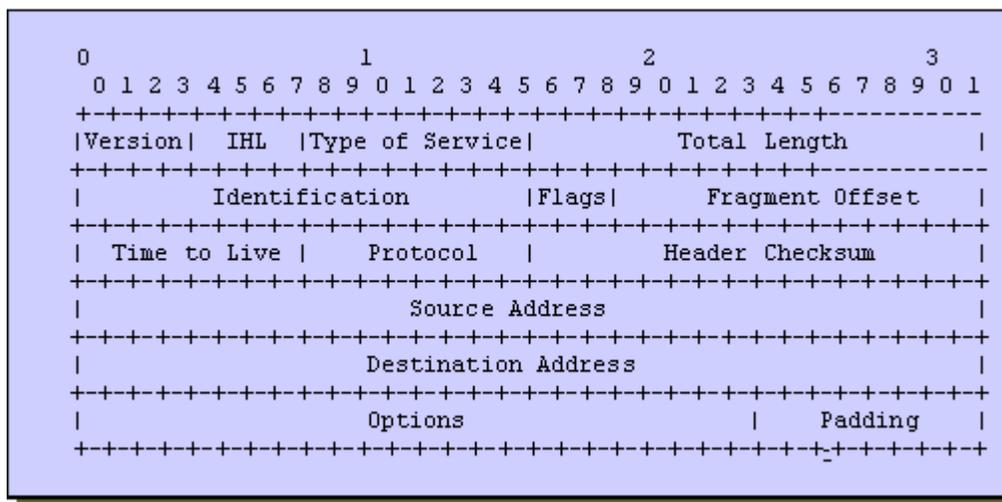


Figure 1: IP Header

The first field we want to look at is the Type of Service field (TOS). The TOS field conveys the operating system and tells if there are any special requirements for the data. This field is like a quality of service field. For example, some operating systems will use the TOS setting 0x10 (minimize delay) for initializing telnet sessions. This can aid an analyst in identifying an operating system. By default many operating systems set their TOS fields to 00 (normal).

There has been a little confusion on the streets now about a new standard called Explicit Congestion Notification (ECN). This standard has changed the shape of our TCP/IP headers. In the past the TOS field was pretty simple, you had a precedence field (3-bits), a TOS field (4-bits) and a Must be Zero (MBZ) field (1 bit) for a total of 8 bits (1 byte). In the TOS service field a computer could set one of the following bits:

- 1000 - Minimize Delay
- 0100 - Maximize Throughput
- 0010 - Maximize Reliability
- 0001 - Minimize monetary cost
- 0000 - Normal Service

ECN has started to be deployed beginning with Linux 2.4. With that the TOS field changes. The following is from my ECN and it's impact on intrusion detection paper:

Two bits in the IP header that will be used are bits 6(left of the low order bit) and 7(low order bit) in the TOS field. Over the years, bits 6 and 7 have had various uses and meanings. Most recently, bit 6 was used by the TOS field and was defined as "Minimize Monetary Cost" with a hex value of 0x02. Typically, NNTP would use the TOS value 0x02, however this function became obsolete by the "Differentiated Services" field for IPv4 and IPv6. Bit 7 was set to MBZ (must be zero). Again, this function became obsolete by the "Differentiated Services" field for IPv4 and IPv6. As part of ECN, TOS bit 6 will now be used as an ECN capable transport (ECT). This bit will be set by the sender stating that the end points are ECN compatible [1]. Bit 7 will now be used as a Congestion Experienced bit (CE). This bit is set by routers that detect congestion on the network using Random Early Detection (RED). Through my experience though, I have not seen Red Hat Linux 7.1 use ECN

in a SYN packet. This can be done if you recompile your kernel.

For more information on TOS and the changes it has gone through over the years please see <http://www.rfc-editor.org/rfc/rfc2474.txt>. If you need more info on ECN please see <http://www.rfc-editor.org/rfc/rfc3168.txt>.

The total length field tells us how long the whole packet is. This includes the embedded header, IP header, and the payload. When I originally sent this paper out to be reviewed I had some discussions on the total packet length and its impact on not only passive OS fingerprinting but also intrusion detection. How? The total length of a packet is very dependent on the operating system (especially a SYN packet). Each operating system sets its own TCP options along with nop's, these options affect the total length and therefore make each OS unique in total packet length. An operating system can be identified in some cases just by the packet length alone on the SYN packet. Also, when looking at TCP packets keep in mind that almost all operating systems use at least one TCP option on the SYN packet. Normally, this is the MSS option and that option is 4 bytes in length (making the minimum total length = 44 bytes). Therefore, if you are seeing a 40 byte packet coming in or out of your network... I would be hunting down the source because it is a crafted packet.

The third field we want to pay attention to is the IP ID field. This field is used mainly in fragmentation but can play a big role in passive OS identification. We will cover this field in much more detail later in this paper.

The Time-to-live (TTL) tells us how long a packet can stay on the wire. It is decremented by one for each hop (router) that the packet passes through. As you will see later in this paper, an initial TTL value can say a lot about an operating system.

There are a lot of references out there that state that the DF field is a critical part in passive OS fingerprinting. Well, the DF field can be used to help identify an operating system but the problem is...most operating systems set the DF flag by default; therefore, this field becomes useless in identifying many operating systems.

Now that we have covered IP, let us look at the Transmission Control Protocol (TCP) and its header. Figure 2 identifies what a TCP header looks like.

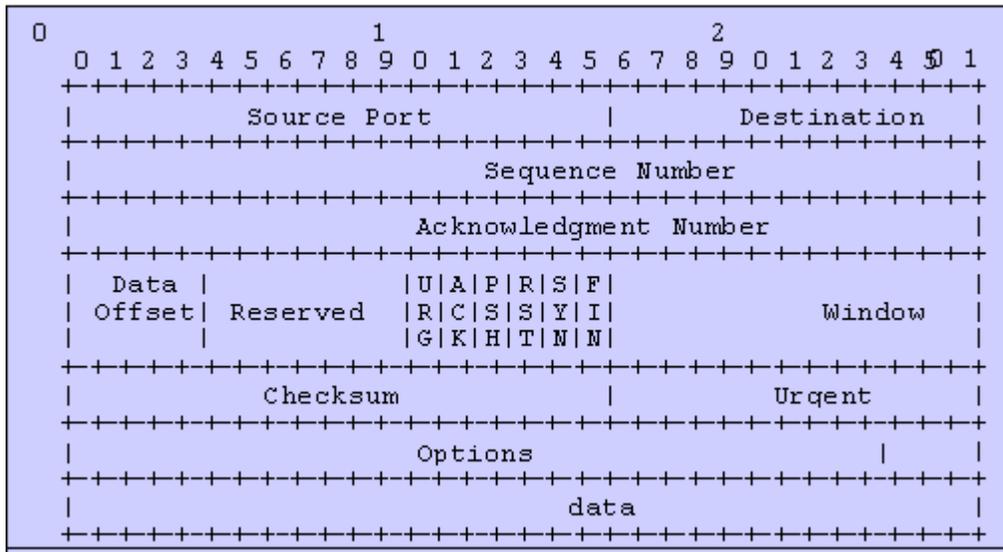


Figure 2: TCP Header

In the TCP header, we will want to begin by focusing on the source port field. This field can tell us some details about an operating system, but not enough to accurately identify the operating system. For example, a RedHat Linux system by default uses source ports that range from 1024 through 4999. However, if you are using NAT then you can throw that little piece of knowledge right out of the door. Why? This is because NAT will use high source ports.

Let us now look at the window field. This field tells us how many octets of data the sending computer is ready to receive. Keep in mind that once a connection has been made, this number can change depending on how much data needs to be passed. The initial window value can be different from operating system to operating system.

Finally, let us look at the most important field in the TCP header when it comes to passive operating system fingerprinting. This field is the *TCP options field*. This field by itself can almost identify most operating systems. Keep in mind that all of the operating systems I have researched use at least one TCP option in its SYN packet. Here is a list of the available options:

- Maximum Segment Size (MSS) - This is normally your MTU - 40.
- Timestamp - The main function for timestamps is to help measure delays.
- Window Scale (wscale) - RFC 1323 states the window scale function as:

"It has two purposes: (1) indicate that the TCP is prepared to do both send and receive window scaling, and (2) communicate a scale factor to be applied to its receive window." [2]

- Selective Acknowledgement (SackOK) - Selective Acknowledgement is used to inform the sender of all received data so that the sender can re-transmit only the data that has not been received yet.
- Nop : nop's are 1 byte in size and are used to pad the TCP options field to number that can be divided by 4.

Keep these options in mind, as we will see many of these options a little later in this paper.

1.3 OPERATING SYSTEMS

During my research, I tested the following operating systems:

- Linux 2.4, 2.2
- OpenBSD 2.9
- Solaris 7
- Aix 4.3
- Windows 2000

Let's look at some of the operating systems and try to identify them using passive techniques.

1.3.1 LINUX

Linux 2.4 and 2.2 are interesting operating systems. Lets begin with a quick look at a TCPDUMP trace of a Linux 2.4 SYN packet:

```
17:05:33.528418 192.168.1.5.32771 > 192.168.1.55.telnet: S [tcp sum ok]
2597834576:2597834576(0) win 5840 <mss 1460,sackOK,timestamp 27205
0,nop,wscale 0> (DF) (ttl 64, id 42261, len 60)
      4500 003c a515 4000 4006 121a c0a8 0105
      c0a8 0137 8003 0017 9ad7 cf50 0000 0000
      a002 16d0 5922 0000 0204 05b4 0402 080a
      0000 6a45 0000 0000 0103 0300
```

Figure 3: Linux 2.4 Packet Trace

Figure 3 is an excellent example of how a SYN packet can be a gold mine of information, when it comes to passive OS fingerprinting. Let us begin looking at the Window size. As we can see in Figure 3, Linux 2.4 sets its initial Window size to 5840 bytes. This is different from a Linux 2.2 kernel, which normally sets its initial window size to 32120.

Our next area to look at in this packet is the TTL. The TTL is great but it cannot identify an operating system by itself. By default, Linux kernel 2.2 and 2.4 use a default TTL value of 64. You can verify that value by issuing the following:

```
More /proc/sys/net/ipv4/ip_default_ttl
```

The IP ID is our next area of interest. Linux by default increments its IP ID by one during each session. BUT, it uses random IP IDs at the start of each new session. Let us look at some packets and see what I mean.

```
17:05:30.773757 192.168.1.5.32770 > 192.168.1.55.telnet: S [tcp sum ok]
2598191518:2598191518(0) win 5840 <mss 1460,sackOK,timestamp 26929
0,nop,wscale 0> (DF) (ttl 64, id 10415, len 60)
      4500 003c 28af 4000 4006 8e80 c0a8 0105
      c0a8 0137 8002 0017 9add 419e 0000 0000
      a002 16d0 e7e3 0000 0204 05b4 0402 080a
      0000 6931 0000 0000 0103 0300
17:05:30.777554 192.168.1.55.telnet > 192.168.1.5.32770: R [tcp sum ok]
0:0(0) ack 2598191519 win 0 (DF) (ttl 64, id 17807, len 40)
      4500 0028 458f 4000 4006 71b4 c0a8 0137
      c0a8 0105 0017 8002 0000 0000 9add 419f
      5014 0000 cfad 0000
17:05:33.528418 192.168.1.5.32771 > 192.168.1.55.telnet: S [tcp sum ok]
2597834576:2597834576(0) win 5840 <mss 1460,sackOK,timestamp 27205
0,nop,wscale 0> (DF) (ttl 64, id 42261, len 60)
      4500 003c a515 4000 4006 121a c0a8 0105
      c0a8 0137 8003 0017 9ad7 cf50 0000 0000
      a002 16d0 5922 0000 0204 05b4 0402 080a
      0000 6a45 0000 0000 0103 0300
17:05:33.528832 192.168.1.55.telnet > 192.168.1.5.32771: R [tcp sum ok]
0:0(0) ack 2597834577 win 0 (DF) (ttl 64, id 7369, len 40)
      4500 0028 1cc9 4000 4006 9a7a c0a8 0137
      c0a8 0105 0017 8003 0000 0000 9ad7 cf51
      5014 0000 4200 0000
```

Figure 4: Linux Traffic

Take a look at the first SYN packet (attempted session), there we see an IP ID of 10415. If we look at the next SYN packet, we see that the IP ID is 42261. As I stated earlier, if a connection is not made then the IP ID's are random. However, once a connection has been established the IP ID that was used in the SYN packet will then be incremented by one during that session. Now, this trait makes Linux 2.4 interesting.

Our next field to look at is the TCP Options field. Take a look at Figure 5 and see what options TCP uses.

```
17:05:30.773757 192.168.1.5.32770 > 192.168.1.55.telnet: S [tcp sum ok]
2598191518:2598191518(0) win 5840 <mss 1460,sackOK,timestamp 26929
0,nop,wscale 0> (DF) (ttl 64, id 10415, len 60)
      4500 003c 28af 4000 4006 8e80 c0a8 0105
      c0a8 0137 8002 0017 9add 419e 0000 0000
      a002 16d0 e7e3 0000 0204 05b4 0402 080a
      0000 6931 0000 0000 0103 0300
```

Figure 5: TCP Options

Figure 5 shows us a typical Linux 2.4 packet. If you look at the TCP Options (in bold), you can see that by default Linux 2.4 and 2.2 set the following TCP Options:

- Maximum Segment Size (MSS)
- Selective Acknowledgment OK (sackOK)
- Timestamp
- Window Scale
- nop

This is very important. Why? Because the TCP Options that Linux (or any OS for that matter) uses, plays a huge role in how big the total packet length is.

Since we brought up Total Length of a packet lets look at the Total Length of a typical Linux 2.4 packet. If you look at Figure 5, we see that the total length (in green) of the packet is 60 bytes. This is important, because Linux is the only operating system that I am aware of that has a total header length of sixty (60) bytes. Because of this trait, Linux is easy to identify.

1.3.1.1 Putting Linux Fingerprinting Together

In the previous section we covered all of the default settings used in a Linux 2.4 stack and those settings that can be used to identify Linux. Now we want to put all of that information together and start identifying Linux "in the wild".

```
17:05:26.310940 192.168.1.5.32769 > 192.0.0.168.telnet: S [tcp sum ok]
2587728312:2587728312(0) win 5840 <mss 1460,sackOK,timestamp 26483
0,nop,wscale 0> (DF) (ttl 64, id 65227, len 60)
      4500 003c fecb 4000 4006 b99a c0a8 0105
      c000 00a8 8001 0017 9a3d 99b8 0000 0000
      a002 16d0 935f 0000 0204 05b4 0402 080a
      0000 6773 0000 0000 0103 0300
```

Figure 6: Linux packet

Looking at Figure 6, we see a few things that stand out.

- TTL:64
- Windows: 5840
- TCP Options: Sets MSS, Timestamps, sackOK, wscale and 1 nop
- Packet Length:60

Before we continue, I want to point out that the reason I did not mention the IP ID is because we would need to see more than just a single packet to be able to use the IP ID. If you look at all of these characteristics, they all point toward one operating system. That operating system is Linux.

1.3.2 OPENBSD

The second operating system we will look at is OpenBSD. OpenBSD is considered the most secure operating system on the market today. But, it also has a few characteristics that make it easy to identify from a passive operating system perspective.

Lets begin this lesson by looking at Figure 7.

```

15:04:46.254692 < 192.168.1.55.23109 > 192.168.1.5.ssh: S
3609264599:3609264599(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,[|tcp]> (DF) (ttl 64, id 32923)
          4500 0040 809b 4000 4006 3690 c0a8 0137
          c0a8 0105 5a45 0016 d721 01d7 0000 0000
          b002 4000 6d31 0000 0204 05b4 0101 0402
          0103 0300 0101

```

Figure 7: OpenBSD Packet

What makes OpenBSD easy to detect? Lets look at a few key indicators:

- TTL: 64. This value is used by many other operating systems as well. You will not be able to detect OpenBSD based on this value alone.
- Window Size: 16384. This value is used by other operating systems as well.
- TCP Options: Uses the same options as Linux BUT instead of setting one nop, OpenBSD uses 5 nops
- IP ID: Totally random. Will need more than one packet to use effectively.
- Total Packet Length: 64 Bytes. This is a key indicator.

As I indicated earlier, OpenBSD is not that hard to identify. If you have 2 or 3 of the characteristics listed, you can identify OpenBSD with relative ease.

1.3.3 SOLARIS 7

Solaris is the next operating system we will look at. By default, Solaris has one characteristic that makes it easy to identify. That characteristic is the initial *TTL field*.

```

09:57:19.031944 < 192.168.0.25.32841 > 192.168.0.1.telnet: S
83052643:83052643(0) win 8760 <mss 1460> (DF) (ttl 255, id 53490)
          4500 002c d0f2 4000 ff06 296e c0a8 0019
          c0a8 0001 8049 0017 04f3 4863 0000 0000
          6002 2238 26cd 0000 0204 05b4 5555

```

Figure 8: Solaris 7 Packet

Look at Figure 8 to see what kind of vital information we can gather from this packet:

- TTL: 255
- Window Size: 8760
- TCP Options: MSS
- IP ID: Increments by one ALL of the time.
- Total Packet Length: 44 bytes

What makes this packet a Solaris packet? If you combine the fact that the TTL for Solaris is 255 along with the fact that the packet length is 44 bytes, you can safely bet that the operating system being used is Solaris. Before I go on, I need to emphasize that there are a few operating systems that also have a total length of 44 bytes. They do not have a TTL of 255. Another unique characteristic with Solaris is the IP ID. Solaris along with AIX 4.3 and Windows 2000 all increment the IP ID's by one ALL of the time. There is no randomness involved when it comes to the IP ID's and Solaris.

1.3.4 AIX 4.3

AIX has some similarities to Solaris and one to OpenBSD | Linux.

```

08:15:55.012972 192.168.0.60.57551 > 192.168.0.1.telnet: S [tcp sum ok]
1677269879:1677269879(0) win 16384 <mss 1460> [tos 0x10] (ttl 60, id
51280)
          4510 002c c850 0000 3c06 34de c0a8 003c
          c0a8 0001 e0cf 0017 63f9 1b77 0000 0000
          6002 4000 7641 0000 0204 05b4 0000

```

Figure 9: AIX packet

By looking at Figure 9 we can come up with these conclusions:

- TTL: 64. Resembles Linux and OpenBSD
- Window: 16384. Resembles Windows 2000 (covered next)
- TCP Options: MSS. Like Solaris

- Packet Length: 44 bytes. Again, like Solaris.
- IP ID: Increments by one ALL of the time

Detecting AIX requires a little work, but it is not impossible. One has to look at more than one field to be able to identify an AIX 4.3 system.

1.3.5 WINDOWS 2000

Our final operating system we will be looking at is Windows 2000. Windows 2000 is really not that hard as well to identify. Lets look at a Windows 2000 packet:

```
05:52:29.715325 < 192.168.0.1.1040 > 192.168.0.55.telnet: S
3595397846:3595397846(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl
128, id 37348)
                                4500 0030 91e4 4000 8006 e75a c0a8 0001
                                c0a8 0037 0410 0017 d64d 6ad6 0000 0000
                                7002 4000 7c4b 0000 0204 05b4 0101 0402
```

Figure 10: Windows 2000 Packet

By looking at Figure 10 we can come up with these conclusions:

- TTL: 128. Almost all Windows Operating systems use this TTL.
- Window: 16384 - just like AIX 4.3
- TCP Options: MSS, sackOK, 2 nops
- Packet Length: 48 bytes. Unique among the operating systems that I have tested.
- IP ID: Increments by one ALL of the time

Detecting Windows 2000 is not hard. For the most part, you could detect a Windows 2000 machine based on the packet length by itself. If you add in the TCP Options as well as the TTL you have indicators that are pretty accurate.

1.4 CONTINUING RESEARCH

Until now we have focused on a plain Jane SYN packet, you know the basics. During this study I found some other interesting characteristics as well.

First, when testing these operating systems I wanted to see if using certain applications (i.e. telnet) could change the SYN. I tried FTP and nothing interesting came up, I tried telnet and I found two operating systems that set their TOS minimize delay bit during the initial SYN. Those two operating systems are OpenBSD and AIX 4.3. Figure 11 is an OpenBSD packet:

```
15:07:28.824692 < 192.168.1.55.5007 > 192.168.1.5.telnet: S
3208525575:3208525575(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,[|tcp]> (DF) [tos 0x10] (ttl 64, id 63852)
                                4510 0040 f96c 4000 4006 bdae c0a8 0137
                                c0a8 0105 138f 0017 bf3e 3707 0000 0000
                                b002 4000 9554 0000 0204 05b4 0101 0402
                                0103 0300 0101
```

Figure 11: OpenBSD 2.9 Packet

```
08:15:55.012972 192.168.0.60.57551 > 192.168.0.1.telnet: S [tcp sum ok]
1677269879:1677269879(0) win 16384 <mss 1460> [tos 0x10] (ttl 60, id
51280)
                                4510 002c c850 0000 3c06 34de c0a8 003c
                                c0a8 0001 e0cf 0017 63f9 1b77 0000 0000
                                6002 4000 7641 0000 0204 05b4 0000
```

Figure 12: AIX 4.3 Packet

In both figures we see that both AIX and OpenBSD set the minimize delay bit during the SYN. Ok, what's the big deal. Well, most operating system I have seen usually do not set TOS bits until after the TCP connection has been established. Knowing this we now have another piece of the pie to put together in passive OS fingerprinting.

During this research time, I began to sit back and wonder how we could identify operating systems using packets that do not contain a SYN flag. I began looking into this and found an away to identify AIX, Solaris and Microsoft Windows. Linux and OpenBSD can be identified but may require more than one packet to do so.

In order to do this we should break out a handy dandy PUSH | ACK packet and look at it:

```
17:28:26.866149 192.168.1.5.32774 > 216.239.37.101.http: P 1:306(305)
ack 1 win 5840 <nop,nop,timestamp 164448 456985791> (DF) (ttl 64, id
55504, len 357)
      4500 0165 d8d0 4000 4006 a0c0 c0a8 0105
      d8ef 2565 8006 0050 f057 5a91 d7b8 4645
      8018 16d0 9b40 0000 0101 080a 0002 8260
      1b3d 0cbf 4745 5420 2f20 4854 5450 2f31
      2e31 0d0a 436f 6f6b 6965 3a20 5052 4546
      3d49
```

Figure 13: PUSH | ACK packet

Lets take a closer look and see what's going on. First, of all we see that 192.168.1.5 is pushing 305 bytes of data across the wire. No big deal. Well, next lets look at the total length of this packet. This packets total length is 357 bytes. If you do the math and subtract 305 from 357, you will get 52. Linux and OpenBSD both have a header size of 52 bytes when pushing data across. How did I come up with 52 bytes? Well, look at the TCP Options in Figure 13. You can see here that this packet has 2 nop's set along with the timestamp option set. The Timestamp is 10 bytes in length and the nop's are one byte each in length for a total length of 12 bytes. The problem with Linux and OpenBSD is that both use the same TTL value. So in order to identify the two we would need to look at the IP ID. This would require us to look at more than one packet. Remember OpenBSD's IP ID are totally random. Therefore if we were to see random IP ID's during a session we could say that this is an OpenBSD box. Otherwise we are left saying that this packet is a Linux packet. (BTW, for FYI... this is a Linux machine.)

```
16:57:59.119669 192.168.1.250.1201 > www.linuxsecurity.com.http: P [tcp
sum ok] 1:419(418) ack 1 win 17520 (DF) (ttl 128, id 3392, len 458)
0x0000 4500 01ca 0d40 4000 8006 ed35 c0a8 01fa E....@....5....
0x0010 d10b 6b0a 04b1 0050 ce4a 32c1 73bf b4d0 ..k....P.J2.s...
0x0020 5018 4470 c022 0000 4745 5420 2f20 4854 P.Dp..."GET./.HT
0x0030 5450 2f31 2e31 0d0a 486f 7374 3a20
```

Figure 14: More PUSH | ACK packets

The packet shown in Figure 14 is different then the packet in Figure 13. How? Well, subtract the data being pushed from the total length. You will see a header length of 40 bytes. No TCP options, no nothing. This is typical of Windows 2000, Solaris 7 and AIX.

Finally, we have Red Hat Linux 7.1 and 7.2. Yes, I know that we have covered Linux but not this flavor of Linux. When a person installs Red Hat 7.1 or 7.2 he/she is given an option of setting up a default firewall. No big deal right? Wrong! Because of how Red Hat sets up the firewall, it provides us with a great deal of information to perform passive OS fingerprinting.

When installing a default load of Red Hat 7.1 or 7.2 the user is asked if he/she wants to set up a firewall. If the user is in a hurry and selects *NEXT*, then the default selection is *Medium Security*. This is bad! When set to Medium Security; Red Hat protects all resources below port 1023. Also, it does not allow access to the NFS Server (port 2049), X-Windows system (port 6000) and the X-font server system. What about ports 1024 -65535? Don't these ports count? Guess not. So, can you guess which firewall Red Hat chooses to use? I will help here. It's not IPTABLES. You guessed it correctly! It's IPCHAINS. Why? Well, that is a story for another paper. I bring this configuration up because instead of dropping the packets (like a good firewall should), Red Hat rejects the packets (using ICMP packets). This tells us a lot about Red Hat 7.1 and 7.2. Figure 15 shows us what I have been discussing:

```
15:04:46.254692 < 192.168.1.55.23109 > 192.168.1.5.ssh: S
3609264599:3609264599(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,[tcp]> (DF) (ttl 64, id 32923)
      4500 0040 809b 4000 4006 3690 c0a8 0137
      c0a8 0105 5a45 0016 d721 01d7 0000 0000
      b002 4000 6d31 0000 0204 05b4 0101 0402
      0103 0300 0101
15:04:46.254692 < 192.168.1.5 > 192.168.1.55: icmp: 192.168.1.5 tcp port
ssh unreachable Offending pkt: [[tcp] (DF) (ttl 64, id 32923) (DF) [tos
0xc0] (ttl 255, id 0)
      45c0 005c 0000 4000 ff01 f753 c0a8 0105
      c0a8 0137 0303 80bc 0000 0000 4500 0040
      809b 4000 4006 3690 c0a8 0137 c0a8 0105
      5a45 0016 d721
```

Figure 15: Red Hat replies

In Figure 15, we see two packets. The first is a typical SYN packet that we have come to know and love. The second should be a RST or nothing at all (if using a firewall) but instead we see an ICMP TCP port SSH unreachable. How can we use this for passive OS fingerprinting? There are a couple of ways to use this. First, if you were sniffing and saw these replies only on ports below 1023 and RST's on ports above 1023 then you are probably dealing with a Red Hat machine with a default install. Second, the packet size for the ICMP

packet is 92 bytes. During my research I have found that this is pretty consistent with this type of packet. Because of the default firewall setup that Red Hat uses we can easily identify a default Red Hat install on either a 7.1 machine or 7.2 machine.^[3]

1.5 CONCLUSION

Although, I have not included all operating systems in this paper, this should be a good start. If we can perfect this technique we can do many things with the information (such as threat analysis). Passive OS fingerprinting is a great tool to use but keep in mind that this technique is not 100% spoof proof. Yet, most traffic that you monitor probably will not be crafted to taint the fields used for passive OS fingerprinting. So, it remains a very powerful method of identifying operating systems.

[1] <http://www.rfc-editor.org/rfc/rfc2481.txt>

[2] <http://www.rfc-editor.org/rfc/rfc1323.txt>

[3] I would like to thank Ofir Arkin and Jay Beale in helping me with the research and discussions dealing with Red Hat 7.1 and 7.2 default firewall issue.