Passive OS fingerprinting: Details and Techniques (Part 2)

Purpose

The purpose of this paper is to provide a detailed explanation of passive OS fingerprinting. This paper will briefly cover the current techniques for identifying operating systems, and new techniques in identifying operating systems.

The beginning

Passive OS fingerprinting is the technique in which people look at packets being sent to them and identify the system it came from. This technique is a few years old and several papers have been written about this technique. If your interested in reading up on passive OS fingerprinting before diving into this paper then check out my article on passive OS fingerprinting. Let's begin by looking at some packets and performing some passive OS fingerprinting:

```
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 1: **Packet Trace**

The first field I typically look at when performing passive OS fingerprinting is the total length field in the IP header. In Figure 1, the total length field is shown to have a value of 3c. In case you don't have a calculator handy, 3c comes out to 60 bytes in decimal. Ok, what does that tell us? For now, it doesn't really tell us anything other than there are at least two operating systems that have a total length value of 60 bytes. So now we have to begin looking at the TCP options field in the packet. In Figure 1, we see that this operating system sets a couple of options. It sets the *Maximum Segment Size* value to 1460. It also sets the Selective Acknowledgement "OK flag" as well as the timestamp and wscale options. And finally, it also uses nop's to pad the value of the options field.

If we had more then one packet, we could look at the IP id field in order to help us identify this operating system. But in this case, that field does not help us at all. So, there are only 2 fields that will help us in making this decision. The first field is the Time-to-Live field, in this case our value is 64; our second field in this case is the Window size field and its value is 5840. Ok, lets perform some passive OS fingerprinting math and see what we come up with:

TTL = 64 + Window size = 5840 + TCP Options = 1 nop, MSS, Wscale, timestamp and SackOK + Total Length = 60 = Linux 2.4 operating system.

New Techniques

Passive OS fingerprinting techniques like I have described above have been around for a couple of years and are used by passive OS fingerprinting tools through out our community. This section of this paper will cover other techniques we can use in order to identify operating systems…passively.

How many times have you been monitoring TCPdump or something similar and have seen 3-12 SYN's coming at you and wondered why? The technique I have began using to perform passive OS fingerprinting will help explain why?

Red Hat Linux

Let's begin by looking at a Linux machine first. Hopefully, everyone is familiar with the /proc directory. If your not, the /proc directory is pseudo like file system that allows us to view many different areas of /dev/kmem. The area we are mainly interested in when performing passive OS fingerprinting is /proc/sys/net/ipv4. Lets perform an ls –la and see what we come up with (truncated for space):

```
tcp_abort_on_overflow
tcp_adv_win_scale
tcp_app_win
tcp_dsack
tcp_ecn
tcp_fack
tcp_fin_timeout
tcp_keepalive_intvl
tcp_keepalive_probes
tcp_keepalive_time
tcp_max_orphans
tcp_max_syn_backlog
tcp_max_tw_buckets
tcp_mem
tcp_orphan_retries
tcp_reordering
tcp_retrans_collapse
tcp_retries1
tcp_retries2
tcp_rfc1337
tcp_rmem
tcp_sack
tcp_stdurg
tcp_synack_retries
tcp_syncookies
tcp_syn_retries
tcp_timestamps
tcp_tw_recycle
tcp_window_scaling
tcp_wmem
```

Figure 2: TCP results from the /proc directory

Although I have cut a lot of the ls-la results out of the paper, we can see that there are still many interesting results available to us to look at. The one we want to look at for this paper is the tcp_syn_retries. What is tcp_syn_retries? Well, it's quite simple, tcp_syn_retries is the number of times the operating system will try make a session if the first attempts fails. Lets take a look at a Red Hat machine re-attempting to make a connection:

```
20:58:36.804058 10.10.10.100.1030 > 10.10.10.25.http: S [tcp sum ok] 287415246:287415246(0) win 5840
<mss 1460,sackOK,timestamp 99459477 0,nop,wscale 0> (DF) (ttl 64, id 14939, len 60)
0x0000       4500 003c 3a5b 4000 4006 d7d0 0a0a 0a64        E..<:[@.@......d
0x0010       0a0a 0a19 0406 0050 1121 9bce 0000 0000        .......P.!......
0x0020       a002 16d0 afde 0000 0204 05b4 0402 080a        ................
0x0030       05ed a195 0000 0000 0103 0300                  ............
20:58:39.794283 10.10.10.100.1030 > 10.10.10.25.http: S [tcp sum ok] 287415246:287415246(0) win 5840
<mss 1460,sackOK,timestamp 99459777 0,nop,wscale 0> (DF) (ttl 64, id 14940, len 60)
0x0000       4500 003c 3a5c 4000 4006 d7cf 0a0a 0a64        E..<:\@.@......d
0x0010       0a0a 0a19 0406 0050 1121 9bce 0000 0000        .......P.!......
0x0020       a002 16d0 aeb2 0000 0204 05b4 0402 080a        ................
0x0030       05ed a2c1 0000 0000 0103 0300                  ............
20:58:45.794266 10.10.10.100.1030 > 10.10.10.25.http: S [tcp sum ok] 287415246:287415246(0) win 5840
<mss 1460,sackOK,timestamp 99460377 0,nop,wscale 0> (DF) (ttl 64, id 14941, len 60)
0x0000       4500 003c 3a5d 4000 4006 d7ce 0a0a 0a64        E..<:]@.@......d
0x0010       0a0a 0a19 0406 0050 1121 9bce 0000 0000        .......P.!......
0x0020       a002 16d0 ac5a 0000 0204 05b4 0402 080a        .....Z..........
0x0030       05ed a519 0000 0000 0103 0300                  ............
20:58:57.794307 10.10.10.100.1030 > 10.10.10.25.http: S [tcp sum ok] 287415246:287415246(0) win 5840
<mss 1460,sackOK,timestamp 99461577 0,nop,wscale 0> (DF) (ttl 64, id 14942, len 60)
0x0000       4500 003c 3a5e 4000 4006 d7cd 0a0a 0a64        E..<:^@.@......d
0x0010       0a0a 0a19 0406 0050 1121 9bce 0000 0000        .......P.!......
0x0020       a002 16d0 a7aa 0000 0204 05b4 0402 080a        ................
0x0030       05ed a9c9 0000 0000 0103 0300                  ............
20:59:21.794303 10.10.10.100.1030 > 10.10.10.25.http: S [tcp sum ok] 287415246:287415246(0) win 5840
<mss 1460,sackOK,timestamp 99463977 0,nop,wscale 0> (DF) (ttl 64, id 14943, len 60)
0x0000       4500 003c 3a5f 4000 4006 d7cc 0a0a 0a64        E..<:_@.@......d
0x0010       0a0a 0a19 0406 0050 1121 9bce 0000 0000        .......P.!......
0x0020       a002 16d0 9e4a 0000 0204 05b4 0402 080a        .....J..........
0x0030       05ed b329 0000 0000 0103 0300                  ...)........
```

Figure 3: RedHat Linux SYN retries

Ok, I know what your thinking…so what! There are a couple of items we need to look at in order to help us with the passive OS fingerprinting. First of all, RedHat Linux sends 5 SYNs in order to try and make a connection. This is critical because not all operating systems send the same number of SYN's. The second bit of information we need to look at is the time delay between packets. The time difference between the first packet and the second packet is three (3) seconds. If you compare the difference between the second and third packet you will see a time difference of (6) seconds, keep on looking at the time differences between the packets and you will begin to notice that the difference between packets begin to double (i.e. 3, 6, 12, 24). The third observation about Figure 3 is the IP ID; yes, under normal circumstances (i.e. A RST is sent back) the IP ID is random until a connection is made. In this case that is not true, as you can see the IP ID's in Figure 3 all increment by 1.

Microsoft Windows 2000 Server

As I stated in the last section, most operating systems have different settings when it comes to tcp_syn_retries. Microsoft is no different lets take a look at the SYN packets Microsoft sends out and see how they differ from other Operating Systems:

```
21:18:27.792633 10.10.10.25.1232 > 10.10.10.70.http: S [tcp sum ok] 2275086460:2275086460(0)
win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id 4990, len 48)
0x0000     4500 0030 137e 4000 8006 bed7 0a0a 0a19         E..0.~@.........
0x0010     0a0a 0a46 04d0 0050 879b 107c 0000 0000         ...F...P...|....
0x0020     7002 4000 7d75 0000 0204 05b4 0101 0402         p.@.}u..........
21:18:30.725191 10.10.10.25.1232 > 10.10.10.70.http: S [tcp sum ok] 2275086460:2275086460(0)
win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id 4992, len 48)
0x0000     4500 0030 1380 4000 8006 bed5 0a0a 0a19         E..0..@.........
0x0010     0a0a 0a46 04d0 0050 879b 107c 0000 0000         ...F...P...|....
0x0020     7002 4000 7d75 0000 0204 05b4 0101 0402         p.@.}u..........
21:18:36.733844 10.10.10.25.1232 > 10.10.10.70.http: S [tcp sum ok] 2275086460:2275086460(0)
win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 128, id 4993, len 48)
0x0000     4500 0030 1381 4000 8006 bed4 0a0a 0a19         E..0..@.........
0x0010     0a0a 0a46 04d0 0050 879b 107c 0000 0000         ...F...P...|....
0x0020     7002 4000 7d75 0000 0204 05b4 0101 0402         p.@.}u..........
```

Figure 4: Microsoft SYN Retries

In figure 4 we see that Microsoft sends three (3) SYN's in an attempt to make a connection. This differs from Linux, which sends five (5) SYN's when attempting a connection. There are similarities between these operating systems and the similarities lie in the time differences between packets. Both Linux and Microsoft begin with a time difference of three (3) second and then double the total for each packet sent. If you're interested in looking at Microsoft's registry settings for TCP/IP look at the following registry entry:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters\Interfaces

<u>FreeBSD</u>

I must admit at this point that FreeBSD is probably the most interesting operating system that I have tested. As you will see a little later, FreeBSD does some strange stuff that will make you scratch your head.

Lets begin by looking at the first three (3) packets that FreeBSD sends out:

```
12:08:30.358526 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43201737 0> (DF) (ttl 64, id 622, len 60)
0x0000     4500 003c 026e 4000 4006 2244 0a01 0103         E..<.n@.@."D....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 3087 0000 0204 05b4 0103 0301         ....0...........
0x0030     0101 080a 0293 34c9 0000 0000                   ......4.....
12:08:33.352457 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43202037 0> (DF) (ttl 64, id 623, len 60)
0x0000     4500 003c 026f 4000 4006 2243 0a01 0103         E..<.o@.@."C....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 2f5b 0000 0204 05b4 0103 0301         ..../[..........
0x0030     0101 080a 0293 35f5 0000 0000                   ......5.....
12:08:36.553100 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43202357 0> (DF) (ttl 64, id 624, len 60)
0x0000     4500 003c 0270 4000 4006 2242 0a01 0103         E..<.p@.@."B....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 2e1b 0000 0204 05b4 0103 0301         ................
0x0030     0101 080a 0293 3735 0000 0000                   ......75....
```

Figure 5: FreeBSD packet traces

What makes FreeBSD so interesting? Well, first of all look at the window size. Yes, 65535 is within the specifications but I have not seen all that many operating systems use a window size this big on a default SYN with nothing else to go on. The second item that needs some attention is the wscale option in the TCP options field. FreeBSD sets its value to 1. This means that the scaling value of wscale is set to 65535 x 2 = 131070 bytes.

Also, look at the total length size of the packet. It is 60-bytes, just like Linux. There are however noticeable differences. FreeBSD uses three (3) nop's where as Linux uses only one.
Ok, with all that being said, none of those are as interesting as what I am about to show you with this trace:

```
12:08:30.358526 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43201737 0> (DF) (ttl 64, id 622, len 60)
0x0000     4500 003c 026e 4000 4006 2244 0a01 0103         E..<.n@.@."D....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 3087 0000 0204 05b4 0103 0301         ....0...........
0x0030     0101 080a 0293 34c9 0000 0000                   ......4.....
12:08:33.352457 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43202037 0> (DF) (ttl 64, id 623, len 60)
0x0000     4500 003c 026f 4000 4006 2243 0a01 0103         E..<.o@.@."C....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 2f5b 0000 0204 05b4 0103 0301         ..../[..........
0x0030     0101 080a 0293 35f5 0000 0000                   ......5.....
12:08:36.553100 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 43202357 0> (DF) (ttl 64, id 624, len 60)
0x0000     4500 003c 0270 4000 4006 2242 0a01 0103         E..<.p@.@."B....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     a002 ffff 2e1b 0000 0204 05b4 0103 0301         ................
0x0030     0101 080a 0293 3735 0000 0000                   ......75....
12:08:39.753769 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 625, len 44)
0x0000     4500 002c 0271 4000 4006 2251 0a01 0103         E..,.q@.@."Q....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
12:08:42.954490 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 626, len 44)
0x0000     4500 002c 0272 4000 4006 2250 0a01 0103         E..,.r@.@."P....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
12:08:46.155125 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 627, len 44)
0x0000     4500 002c 0273 4000 4006 224f 0a01 0103         E..,.s@.@."O....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
12:08:52.356462 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 628, len 44)
0x0000     4500 002c 0274 4000 4006 224e 0a01 0103         E..,.t@.@."N....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
12:09:04.559033 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 629, len 44)
0x0000     4500 002c 0275 4000 4006 224d 0a01 0103         E..,.u@.@."M....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
12:09:28.764146 10.1.1.3.49189 > 10.1.1.6.ssh: S [tcp sum ok] 1124518359:1124518359(0) win
65535 <mss 1460> (DF) (ttl 64, id 630, len 44)
0x0000     4500 002c 0276 4000 4006 224c 0a01 0103         E..,.v@.@."L....
0x0010     0a01 0106 c025 0016 4306 c9d7 0000 0000         .....%..C.......
0x0020     6002 ffff b502 0000 0204 05b4 0000              `...........
```

Figure 6: FreeBSD trace packets Part 2

Before this trace we had just looked at the first three (3) packets of FreeBSD when it performs a SYN retry. Lets take a look at all of the packets FreeBSD uses. Ok, what makes FreeBSD really interesting to me is that fact that after the first three packets, FreeBSD goes from sending SYN packets that set the wscale, MSS, timestamp options set along with three (3) noop's to setting only the Maximum Segment Size. This drops the total length size from 60 bytes down to 44 bytes in size. Lets take a look at some other interesting items associated with FreeBSD:

1) The IP ID increments by one

2) Sends out nine (9) packet during SYN retries

3) Time differences between packets differ from any other operating system I have looked at.

OpenBSD

OpenBSD is considered the most secure out of the box operating system in today's computing world; it is also an operating system that is fun to watch from a TCP/IP perspective as well. Next, let us take a look at OpenBSD and how we should be able to identify it in the wild using SYN retries:

```
12:10:31.405200 10.1.1.103.45447 > 10.1.1.6.ssh: S [tcp sum ok]
3466145834:3466145834(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,timestamp 1787771497 0> (DF) (ttl 64, id 40566, len 64)
0x0000     4500 0040 9e76 4000 4006 85d3 0a01 0167         E..@.v@.@......g
0x0010     0a01 0106 b187 0016 ce99 302a 0000 0000         ..........0*....
0x0020     b002 4000 8a38 0000 0204 05b4 0101 0402         ..@..8..........
0x0030     0103 0300 0101 080a 6a8f 3a69 0000 0000         ........j.:i....
12:10:37.402754 10.1.1.103.45447 > 10.1.1.6.ssh: S [tcp sum ok]
3466145834:3466145834(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,timestamp 1787771509 0> (DF) (ttl 64, id 44980, len 64)
0x0000     4500 0040 afb4 4000 4006 7495 0a01 0167         E..@..@.@.t....g
0x0010     0a01 0106 b187 0016 ce99 302a 0000 0000         ..........0*....
0x0020     b002 4000 8a2c 0000 0204 05b4 0101 0402         ..@..,..........
0x0030     0103 0300 0101 080a 6a8f 3a75 0000 0000         ........j.:u....
12:10:49.405955 10.1.1.103.45447 > 10.1.1.6.ssh: S [tcp sum ok]
3466145834:3466145834(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,timestamp 1787771533 0> (DF) (ttl 64, id 39837, len 64)
0x0000     4500 0040 9b9d 4000 4006 88ac 0a01 0167         E..@..@.@......g
0x0010     0a01 0106 b187 0016 ce99 302a 0000 0000         ..........0*....
0x0020     b002 4000 8a14 0000 0204 05b4 0101 0402         ..@.............
0x0030     0103 0300 0101 080a 6a8f 3a8d 0000 0000         ........j.:.....
12:11:13.412286 10.1.1.103.45447 > 10.1.1.6.ssh: S [tcp sum ok]
3466145834:3466145834(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,timestamp 1787771581 0> (DF) (ttl 64, id 57716, len 64)
0x0000     4500 0040 e174 4000 4006 42d5 0a01 0167         E..@.t@.@.B....g
0x0010     0a01 0106 b187 0016 ce99 302a 0000 0000         ..........0*....
0x0020     b002 4000 89e4 0000 0204 05b4 0101 0402         ..@.............
0x0030     0103 0300 0101 080a 6a8f 3abd 0000 0000         ........j.:.....
```

Figure 7: OpenBSD Packet trace

This trace is a typical OpenBSD SYN retry, if you have forgotten some details related to OpenBSD then let me refresh you memory. OpenBSD with a default SYN usually carries a packet size of 64 bytes. It does this by setting 5 nop's, timestamp, wscale, sackOK and the mss options. Btw, it also has a TTL of 64.

The type of OS fingerprinting we are attempting to do today requires us to look at the number of SYN retries; in this case, it is 4. Look at the time difference between packets, there is a 6 second difference between the first packet and the second packet, a 12 second difference between the second and the third and finally there is a 24 second difference between the third and final packet. Now if you combine all of what we now know of OpenBSD we should be able to come to a pretty good conclusion as to the operating system.

Conclusion

In the intrusion detection world we need to be able to understand what "normal" traffic is all about, one way in doing that is by watching traffic and trying to pick up patterns. This is the key when trying to find ways to perform passive OS fingerprinting.