# Packet forensics using TCP

*Don Parker, Mike Sues*,

Most of us who work in the security world have at one time or another looked at the raw output of a firewall, IDS, or other type of security device. What that output invariably leads one to is viewing packets directly for an investigation. Doing packet forensics can be a difficult and time consuming endeavour. Due to this fact, many of us prefer to use convenient tools such as Ethereal to help facilitate our analysis. There is a notable problem with this approach, however.

While packet analyzers such as Ethereal do an excellent job of breaking out the packets contents, there is one thing that it cannot do for you: it cannot help you understand some of the key relationships that certain packet metrics have. Packet metrics such as the TCP sequence and acknowledgement numbers are a good example of this. Case in point, Ethereal will not tell you if there is a missing packet in your packet capture. The only way to know that a packet is missing would be to go through each and every packet with a keen eye on the aforementioned TCP packet metrics.

If this does not immediately seem all that important to you, please understand that it most certainly is. As a consultant, when we approach a computer network breach for a client, we most definitely need to know exactly what has happened. That means being able to account for each and every packet that was sent during an attack. It is most certainly possible that, for some reason, tcpdump or windump dropped some packets, especially if you are in a high bandwidth network. The problem is that you would not know this fact if you were not aware of how to truly do packet forensics. The ability to see this comes from having a deep understanding of just how protocols talk to each other, in this case the TCP protocol.

This article is set to arm you with the knowledge that allows one to approach a packet stream and successfully be able to determine if there are any missing packets. This is imperative in cases where your data set is missing packets that may contain crucial indicators of the breach. You would only know that by doing the analysis shown below.

One aspect we will not deal with in this article is analysis of application layer data. We shall concentrate with trying to arm you with just the knowledge that you require in order to pull off packet forensics. With that said, let's get to it!

## The journey begins

This document is an effort to fully explain the relationship between TCP sequence numbers and acknowledgement numbers. These two numbers play a logical role in the transmission of packets.

Once the TCP/IP three way handshake is done, the following happens. The next packet sent will have an acknowledgement number in it. This acknowledgement number sent will be the next expected TCP sequence number of the other IP address communicating with it. To further clarify this statement, the Source IP address will send an acknowledgement number that is the next expected TCP sequence number of the Destination IP address. This concept, which is often confusing, is further explained below. Please note that the packet will be provided, and the comments on it shall follow.

```
04/10/2005 10:11:46.263921 10.10.10.10.2748 > 192.168.1.1.25: S [tcp sum
ok] 1635370671:1635370671(0) win 8760  (DF) (ttl 112, id 8408, len 48)
0x0000   4500 0030 20d8 4000 7006 a954 0a0a 0a0a        E..0..@.p..T....
0x0010   c0a8 0101 0abc 0019 6179 c6af 0000 0000        ........ay......
0x0020   7002 2238 ed4d 0000 0204 05b4 0101 0402        p."8.M..........
```

The number underlined above is called the "Initial Sequence Number," or the ISN. This always has to exist in the SYN packet, the first step of the TCP/IP three way handshake. The above packet starts with sending a SYN packet to 192.168.1.1 The "mss 1460" we see above relates to the "maximum segment size" and the 1460 is a value measured in bytes, in other words, 1460 bytes. The "mss 1460" means that 10.10.10.10 wants to receive no more than 1460 bytes of data in any one given packet from 192.168.1.1.

The "nop" as seen above is simply a pad instruction to be used for other TCP options which don't fill out a four byte word. Lastly, "sackOK" relates to "selective acknowledgement," and indicates that it can be used for this session. This "sackOK" and "mss" value should only ever be seen during the SYN and SYN/ACK portion of the TCP/IP handshake. They should not appear during the data portion of the session.

Our last metric of "win 8760" relates to the amount of buffer space that the source IP 10.10.10.10 has, and it is measured in bytes. This value is also commonly known as the receive buffer.

We can determine that the above packet is a SYN packet. The reason for this is we can see the "S," but we also know it more definitively by its hex value of 02, as underlined in the hex content. This is known in hexadecimal as 0x.... The 0x signifies that the number which follows is in hexadecimal format.

Now let's discuss this SYN packet and the byte offset that it's found in (in the TCP header) a little bit more.

**TCP flags explained**

The four core protocols are IP, TCP, UDP and ICMP. When counting the bytes in a packet's core protocol headers, you must start counting from zero instead of one, as you might normally do. Contained in the 13th byte offset from zero in the TCP header is where the TCP flags of FIN, SYN, RST, PSH, ACK, URG are found. Each of these flags has a value assigned to it, and whether that value is best known by you in decimal, or hexadecimal, it amounts to the same thing. The flags each have a value assigned to it. This in turn will allow us to write a bit mask against a binary log file, or when collecting a file via tcpdump. Let's look at a diagram to visualize what we are referring to before moving on.

```
CWR     ECE     URG     ACK     PSH     RST     SYN     FIN
128     64      32      16      8       4       2       1
```

This diagram maps out the "decimal" values of the various flags found in the 13th byte offset (once again, counting from zero) in the TCP header. With this information in hand I could easily devise a BPF filter which also includes a bit mask to pull all packets with a specific flag combination set. For example, using the packet above once again, we could write up the following BPF filter and bitmask to display only PSH/ACK packets, in other words packets that contain data.

```
tcpdump -nXvSs 0 ip and host 10.10.10.10 and host 192.168.1.1 and tcp[13] = 24
```

We arrived at the decimal value of 24 as seen in the above filter by adding up the decimal values of both the ACK and PSH flags. This equals 24, hence our need to tell tcpdump to look in the TCP header, specifically the 13th byte offset from zero, see if that byte has a decimal value of 24 then display it for me.

Please bear in mind that one would normally be running this filter against an already logged binary file, as we'll see below. If we were using the above filter I would be telling tcpdump to only log packets with the above noted IP address and a flag combination equalling 24. The

filter below would be one that I would run against an existing binary log file I have called "bleh."

```
tcpdump -r bleh 0 ip and host 10.10.10.10 and host 192.168.1.1 and tcp[13] = 24 > bleh2
```

The file bleh would be an already logged file of a session between the two noted IP addresses. By having an existing binary log file we can apply bpf/bitmask filters against it interactively. By doing the above, we are telling tcpdump to look for the conditions specified above only, and then to write it to an ASCII file called bleh2.

Remember that it is always best to log in binary for the simple reason that you can write an ASCII file from the binary, but you cannot create a binary log file from an ASCII one. Many tools such as Snort and tcpdump will only work with binary log files, and they will simply not accept flat ASCII files for parsing. With this TCP flag concept fully explained lets move on!

### Back to packet forensics

Let's take a look at a second packet:

```
04/10/2005 10:11:46.264343 192.168.1.1.25 > 10.10.10.10.2748: S [tcp sum
ok] 727167800:727167800(0) ack 1635370672 win 24820  (DF) (ttl 62, id 18217, len 48)
0x0000   4500 0030 4729 4000 3e06 b503 c0a8 0101        E..0G)@.>.......
0x0010   0a0a 0a0a 0019 0abc 2b57 b338 6179 c6b0        ........+W.8ay..
0x0020   7012 60f4 cff0 0000 0101 0402 0204 05b4        p.`.............
```

The second step of the handshake is the SYN/ACK. In this packet, we now have the ISN + 1 as evidenced by the number underlined. We also have a TCP sequence number that is underlined: 727167800:727167800.

Let's move on to step three.

```
04/10/2005 10:11:47.381612 10.10.10.10.2748 > 192.168.1.1.25: . [tcp sum
ok] ack 727167801 win 8760 (DF) (ttl 112, id 8426, len 40)
0x0000   4500 0028 20ea 4000 7006 a94a 0a0a 0a0a        E..(..@.p..J....
0x0010   c0a8 0101 0abc 0019 6179 c6b0 2b57 b339        ........ay..+W.9
0x0020   5010 2238 3b71 0000 0000 0000 0000             P."8;q........
```

This packet is the final step in the TCP/IP three way handshake. The ACK is displayed by the "." (next to the destination port number 25). Don't be confused by the "ack" seen above, which is followed by a number, as it does not signify that the ACK flag is set in the 13th byte offset from zero in the TCP header. In reality, that is the "acknowledgement number" plus one, hence the long number following it. This is an important point to remember.

```
04/10/2005 10:11:47.670592 192.168.1.1.25 > 10.10.10.10.2748: P [tcp sum
ok] 727167801:727167830(29) ack 1635370672 win 24820 (DF) (ttl 62, id
18218, len 69)
0x0000   4500 0045 472a 4000 3e06 b4ed c0a8 0101        E..EG*@.>.......
0x0010   0a0a 0a0a 0019 0abc 2b57 b339 6179 c6b0        ........+W.9ay..
0x0020   5018 xxxx xxxx xxxx xxxx xxxx xxxx xxxx        P.`.....xxxxxxxx
0x0030   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx        xxxxxxxxxxxxxxxx
0x0040   xxxx xxxx xx                                  xxxxxx
```

We now have our 192.168 address sending data, as evidenced by the P seen above. Also, set in the 13th byte offset from 0 is the ACK flag. Note that the 13th byte's offset from zero in the TCP header is where all of the flags reside, in other words: SYN, ACK, RST, FIN, URG, PSH. However, we now know that we do not see a visual representation of the ACK flag in the

packet headers above. We do, however, see it in byte 5018. Specifically, the "18" is in hex, which represents 24 in decimal. This 24 adds up to the PSH and ACK flag decimal values combined.

```
04/10/2005 10:11:49.770114 10.10.10.10.2748 > 192.168.1.1.25: . [tcp sum
ok] ack 727167830 win 8731 (DF) (ttl 112, id 8473, len 40)
0x0000   4500 0028 2119 4000 7006 a91b 0a0a 0a0a        E..(!.@.p.......
0x0010   c0a8 0101 0abc 0019 6179 c6b0 2b57 b356        ........ay..+W.V
0x0020   5010 221b 3b71 0000 0000 0000 0000             P.".;q........
```

In this packet, we have the 10.10.10.10 address acknowledging receipt of the data packet sent to it. In other words, the packet directly above it in this case. Once again, the number fields in question have been underlined. By using an acknowledgement number of 727167830, we know that 192.168.1.1 has received all of the data in that packet.

```
04/10/2005 10:11:51.682538 10.10.10.10.2748 > 192.168.1.1.25: P [tcp sum
ok] 1635370672:1635370706(34) ack 727167830 win 8731 (DF) (ttl 112, id
8501, len 74)
0x0000   4500 004a 2135 4000 7006 a8dd 0a0a 0a0a        E..J!5@.p.......
0x0010   c0a8 0101 0abc 0019 6179 c6b0 2b57 b356        ........ay..+W.V
0x0020   5018 221b b622 0000 4845 4c4f 2074 7332        P."..".."HELO.ts2
0x0030   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx  xxxxxxxxxxxxxxxx
0x0040   6c2e 726f 6c2e 7275 0d0a                      l.rol.ru..
```

A key concept to remember here is that the "ack" number of a given packet is the next expected TCP sequence number from the other IP address in question. So one should note that we have the TCP sequence number here matching the "ack" number of the packet, listed two packets above us. This is exactly as it should be. The packet directly above this one is only acknowledging receipt of a packet, and is not as such sending any data, hence the lack of a TCP sequence number. Sending data would necessitate a TCP sequence number. We can see that above. As this packet is sending data, it has both a TCP sequence number and an "ack" number as well. We would once again will expect to see 192.168.1.1 use this packet's "ack" number as the TCP sequence number.

```
04/10/2005 10:11:51.682814 192.168.1.1.25 > 10.10.10.10.2748: . [tcp sum
ok] ack 1635370706 win 24820 (DF) (ttl 62, id 18219, len 40)
0x0000   4500 0028 472b 4000 3e06 b509 c0a8 0101        E..(G+@.>.......
0x0010   0a0a 0a0a 0019 0abc 2b57 b356 6179 c6d2        ........+W.Vay..
0x0020   5010 60f4 fc75 0000 0000 0000 0000             P.`..u........
```

In this packet our 192.168 address is only acknowledging receipt of the packet above it. That is why there is no TCP sequence number there. This is only an Acknowledgement. Be sure to look at the TCP sequence number of the packet above this one. Note the TCP sequence number on the right. This is being acknowledged by our packet's "ack" number. In essence, this packet is saying I have received your data, and processed it.

```
04/10/2005 10:11:51.682916 192.168.1.1.25 > 10.10.10.10.2748: P [tcp sum
ok] 727167830:727167853(23) ack 1635370706 win 24820 (DF) (ttl 62, id
18220, len 63)
0x0000   4500 003f 472c 4000 3e06 b4f1 c0a8 0101        E..?G,@.>.......
0x0010   0a0a 0a0a 0019 0abc 2b57 b356 6179 c6d2        ........+W.Vay..
0x0020   5018 60f4 xxxx xxxx xxxx xxxx xxxx xxxx        P.`.....xxxxxxxx
0x0030   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx  xxxxxxxxxxxxxxxx
```

First at all, in this packet we have both a TCP sequence number and an "ack" number. Therefore we know that this packet is both acknowledging receipt of data, and is also sending

some in return. In this case, our TCP sequence number is based off of the "ack" number seen two packets above us. This is again exactly as it should be. Remember, the "ack" number of the packet is the next expected TCP sequence number of the other IP address's packet. We also have the same "ack" number repeated here. That is fine, and this "ack" number is once again the next expected TCP sequence number from the other IP address (in this case 10.10.10.10)

```
04/10/2005 10:11:54.820932 10.10.10.10.2748 > 192.168.1.1.25: . [tcp sum
ok] ack 727167853 win 8708 (DF) (ttl 112, id 8541, len 40)
0x0000   4500 0028 215d 4000 7006 a8d7 0a0a 0a0a        E..(!]@.p.......
0x0010   c0a8 0101 0abc 0019 6179 c6d2 2b57 b36d        ........ay..+W.m
0x0020   5010 2204 3b4f 0000 0000 0000 0000             P.".;O........
```

This packet is only an acknowledgement packet as evidenced by the "ack." There is no TCP sequence number here because there was no data sent. In essence 10.10.10.10 is simply saying, "thanks 192.168.1.1, I have received your data."

```
04/10/2005 10:11:58.070622 10.10.10.10.2748 > 192.168.1.1.25: P [tcp sum
ok] 1635370706:1635370747(41) ack 727167853 win 8708 (DF) (ttl 112, id
8591, len 81)
0x0000   4500 0051 218f 4000 7006 a87c 0a0a 0a0a        E..Q!.@.p..|....
0x0010   c0a8 0101 0abc 0019 6179 c6d2 2b57 b36d        ........ay..+W.m
0x0020   5018 2204 42ec 0000 4d41 494c 2046 524f        P.".B...MAIL.FRO
0x0030   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx  xxxxxxxxxxxxxxxx
0x0040   xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx  xxxxxxxxxxxxxxxx
0x0050   0a                                             .
```

In this packet we have both a TCP sequence number and a "ack" number. To clarify, once again this packet's TCP sequence number is based on the "ack" number of the packet two packets above this one. This makes sense, as the packet directly above is only acknowledging the receipt of data, and that "ack" number is also seen here in this packet again. You can now see how hard it would be then to actually hijack a session using TCP sequence number prediction.

```
04/10/2005 10:11:58.070913 192.168.1.1.25 > 10.10.10.10.2748: . [tcp sum
ok] ack 1635370747 win 24820 (DF) (ttl 62, id 18221, len 40)
0x0000   4500 0028 472d 4000 3e06 b507 c0a8 0101        E..(G-@.>.......
0x0010   0a0a 0a0a 0019 0abc 2b57 b36d 6179 c6fb        ........+W.may..
0x0020   5010 60f4 fc35 0000 0000 0000 0000             P.`..5........
```

This packet above is once again only acknowledging the receipt of data. We are used to seeing the "S" to denote a SYN packet and a "P" for a PSH packet. Please realize that an "ACK" is denoted merely as a "." In this case, the "." directly after the destination addresses port number.

### Conclusion

This had been a tutorial on the relationship between the TCP Sequence number and the Acknowledgement number. Understanding this will help you if you ever have to do packet forensics, and will generally increase your knowledge of TCP/IP overall. The authors sincerely hope that you enjoyed it and are able to put this knowledge good use in an effort to further secure your networks.

### About the Authors

Don Parker holds the GCIA and GCIH certifications, and specializes in Intrusion Detection and Incident Handling. He also offers services of a specialized nature to clients with high assurance

networks.

Mike Sues is the CEO of Rigel Kent Security & Training services. In addition to providing clients with specialized, high-assurance security services, he offers an extensive curriculum of technical, lab-oriented computer security training.