


Analysis of a raw TCP/IP packet

In this chapter we are going to have a close look at some captured network packets and apply our knowledge from the previous parts of this series on them. You will see, that a lot of bits and bytes in packets will make much more sense.

Setup

To analyze TCP/IP packets, we are going to setup a small test environment. For this, we will use two machines, one as a (web-) server, the other one as client. You could also use only one system, however, then the source and destination IP address would be the same, making the analysis less intuitive.



The webserver is serving a simple HTML website, which will be requests by a browser from the client. We will listen into this communication with the help of [Wireshark](#), a widely-used network protocol analyzer. It doesn't matter on which of those two systems you use Wireshark, as the communication between them is the same. For our convenience, we will hide non relevant packets with a display filter  Wireshark:

```
ip.addr == 10.10.10.1 && tcp.port == 80
```

Capturing packets

After accessing the webserver with our client's browser, we observe all packets going from or to our server on port 80 with the help of Wireshark:

The image shows a Wireshark packet capture window with the filter 'ip.addr == 10.10.10.1 && tcp.port == 80'. The packet list shows 10 packets. Packets 1, 2, and 3 show the TCP three-way handshake: [SYN], [SYN, ACK], and [ACK]. Packet 4 is an HTTP GET request. Packet 5 is the server's response (200 OK). Packets 6, 7, 8, 9, and 10 show the connection termination sequence with [FIN, ACK] and [FIN] flags.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.10.10.2	10.10.10.1	TCP	74	60156 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=
2	0.000238261	10.10.10.1	10.10.10.2	TCP	74	80 → 60156 [SYN, ACK] Seq=0 Ack=1 Win=2896
3	0.000266617	10.10.10.2	10.10.10.1	TCP	66	60156 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=
4	0.000436546	10.10.10.2	10.10.10.1	HTTP	384	GET / HTTP/1.1
5	0.000755587	10.10.10.1	10.10.10.2	TCP	66	80 → 60156 [ACK] Seq=1 Ack=319 Win=30080 L
6	0.001447666	10.10.10.1	10.10.10.2	HTTP	358	HTTP/1.1 200 OK (text/html)
7	0.001459681	10.10.10.2	10.10.10.1	TCP	66	60156 → 80 [ACK] Seq=319 Ack=293 Win=30336
8	4.950239609	10.10.10.1	10.10.10.2	TCP	66	80 → 60156 [FIN, ACK] Seq=293 Ack=319 Win=
9	4.950416360	10.10.10.2	10.10.10.1	TCP	66	60156 → 80 [FIN, ACK] Seq=319 Ack=294 Win=
10	4.950855367	10.10.10.1	10.10.10.2	TCP	66	80 → 60156 [ACK] Seq=294 Ack=320 Win=30080

These are all the packets going back and forth between our client (10.10.10.2) requesting a webpage from our webserver (10.10.10.1). In the first three packets, we can see the previously described TCP three-way handshake ([SYN] , [SYN, ACK] , [ACK]) for the connection establishment.

In packet number 4, we can see a GET request from the client to the server, being acknowledged by the server in packet 5.

Packet 6 transmits the webpage data to the client, who acknowledges the receipt in packet 7.

Starting with packet 8, we can see a typical connection termination. After the webpage is transmitted from the server to the client, the server sends a [FIN, ACK] to signalize the termination of the connection from its side. The client sends a [FIN, ACK] to acknowledge the [FIN] of the server and send a [FIN] on its own. In packet 10 the server acknowledges the [FIN] of the client. The connection is now terminated.

The first packet in detail

We are going to look at the packet number 1 in more detail. For now we will ignore

the first 14 bytes of the packet, but don't worry, we'll come back to them later.

IP segment

Wireshark shows us all the bits and bytes the packet consists of:

0000	00 0c 29 d3 be d6 00 0c 29 e0 c4 af 08 00 45 00	..).).....E.
0010	00 3c 98 83 40 00 40 06 7a 22 0a 0a 0a 02 0a 0a	.<..@.@. z".....
0020	0a 01 ea fc 00 50 a0 73 df e7 00 00 00 00 a0 02P.s
0030	72 10 28 45 00 00 02 04 05 b4 04 02 08 0a 33 b8	r.(E....3.
0040	42 5f 00 00 00 00 01 03 03 07	B_..... ..

Highlighted in blue is the IP segment of the packet. Let's rearrange the visualization to 32 bit per line:

45	00	00	3c
98	83	40	00
40	06	7a	22
0a	0a	0a	02
0a	0a	0a	01

Now let's visualize those bytes even further using our TCP header layout from the previous chapter:

Bit	0	4	8	16	31
Version	4	5	Type of Service	Total Length	
			00	00 3c	
Identification			Flags	Fragment Offset	
98 83			010 ₂	0000000000000 ₂	
Time to Live		Protocol		Header Checksum	
40		06		7a 22	
Source Address					
0a 0a 0a 02 (= 10.10.10.2)					

Destination Address

0a 0a 0a 01 (= 10.10.10.1)

For better readability, the flags and the fragment offset have been converted from hex to binary. Overall, the values seem to be correct. Just to name a few: IP Version 4 was used, the IP header length (IHL) is in fact 5 (5 * 32 bits), the time to live of the packet is set to 40 (64 in decimal) which is a typical value for *nix systems and the source and destination addresses do also look right.

TCP segment

Now that we had a look at the IP part of the packet, let's look at the TCP segment:

0000	00 0c 29 d3 be d6 00 0c	29 e0 c4 af 08 00 45 00	..).).....E.
0010	00 3c 98 83 40 00 40 06	7a 22 0a 0a 0a 02 0a 0a	.<..@.@. z".....
0020	0a 01 ea fc 00 50 a0 73	df e7 00 00 00 00 a0 02	...P.s
0030	72 10 28 45 00 00 02 04	05 b4 04 02 08 0a 33 b8	r.(E....3.
0040	42 5f 00 00 00 00 01 03	03 07	B_..... ..

Utilizing our previous acquired knowledge about TCP we insert the values in our packet blueprint:

Bit

0

16

31

Source Port ea fc (= 60156₁₀)										Destination Port 00 50 (= 80₁₀)										
Sequence Number a0 73 df e7																				
Acknowledgement Number 00 00 00 00																				
Data Offset 1010₂		000₂	N S	C W	E C	U R	A C	P S	R S	S Y	F I	Window Size 72 10								
	R		E	G	K	H	T	N	N											
	0₂		0₂	0₂	0₂	0₂	0₂	0₂	1₂	0₂										
Checksum 28 45										Urgent Pointer 00 00										
Options (var.) + Padding																				

```
02 04 05 b4
04 02 08 0a
33 b8 42 5f
00 00 00 00
01 03 03 07
```

Again, for the data offset, the reserved space and the flags, the hex values were converted to binary. Judging from the source and destination port, the data offset 1010 (10 in decimal, $10 * 32$ bits) and the set SYN flag, this packet looks alright. You may try this approach with other TCP/IP packets you captured with Wireshark. Either take one from your own Wireshark dump or use [these exercise packets](#). Use the IHL to separate the IP and TCP segments. What kind of packets can you identify?

That's it for the analysis of the packets. In the next part, we will [create a packet from the scratch](#) by ourselves, put it on the wire and send it.

TCP/IP packets

Introduction

1 Recap on network layers and protocols

2 Analysis of a raw TCP/IP packet

3 Manually create and send raw TCP/IP packets

4 Creating a SYN port scanner

Ping – Manually create and send ICMP/IP packets

Contact



Twitter: [@inc0x0](#)

© 2024 inc 0x0

