1 Analysis mit Wireshark

Im Rahmen dieser Sektion wollen wir eine weitere Datei analysieren, welche mit Wireshark aufgenommen wurde, dump_http.pcap.

1.1 Welche Objekte wurden vom Klienten via HTTP angefordert? Mithilfe von

```
File > Export Objects > HTTP...
```

kann man direkt einsehen, welche Elemente im Rahmen aller HTTP-Protokollierten Frames, innerhalb des derzeitig geladenen .pcap übertragen wurden. In unserem dump_http.pcap wurden die Elemente text(145 Byte, Frame 6), logo.gif(4445 Byte, Frame 22) und TechnikErleben.png(27 kB, Frame 66) angefordert, von 192.168.1.10.

1.2 Recherchieren Sie die Bedeutung der einzelnen Header-Felder bei den Anfragen bzw. Antworten des Servers. TCP-Header bestimmen eine ganze Reihe an diversen Datensätzen. Über 10 Felder von je 20 bytes & zusätzlichen 40 byte an optionalen, informativen Daten, bestimmt man

Header des HTTP-Protokolls
Prinzipiell sind die einzelnen HTTP-header ziemlich selbsterklärend. Dennoch für einen kurzen Überblick:
Header bei den Requests:
∘ 'GET ...', ähnlich POST, beschreibt ein Document Request. ∘ 'Accept' besagt ob der User Zugriffsrecht hat

Header bei den Antworten:
∘ 'HTTP/1.1 200 OK' – Beschreibt das genutzte HTTP-Protokoll mitsamt Statuscode, hier OK.
∘ 'Date' beschreibt das Datum an welchem Tag die Nachricht gesendet wurde.
∘ 'Server' beschreibt die genutzte Webserverarchitektur
∘ 'Last-Modified' beschreibt wann die Datei zuletzt verändert wurde.
∘ 'ETag' beschreibt die Version einer Datei und dient dem Caching.
∘ 'Accept-Ranges' ist die Einheit die der Server akzeptiert.
∘ 'Content-Length' beschreibt ¡Body¿ in Byte.
∘ 'Vary' beschreibt ob eine Datei tatsächlich gecached wird oder neu beantragt werden soll.
∘ 'Connection' bezieht sich auf die Folgeaktion. Close steht für das Ende der Session. ∘ 'Content-Type' beschreibt die übertragene Datei.

Header des TCP-Protokolls
◇ Sender TCP Port Nummer (2 Byte)

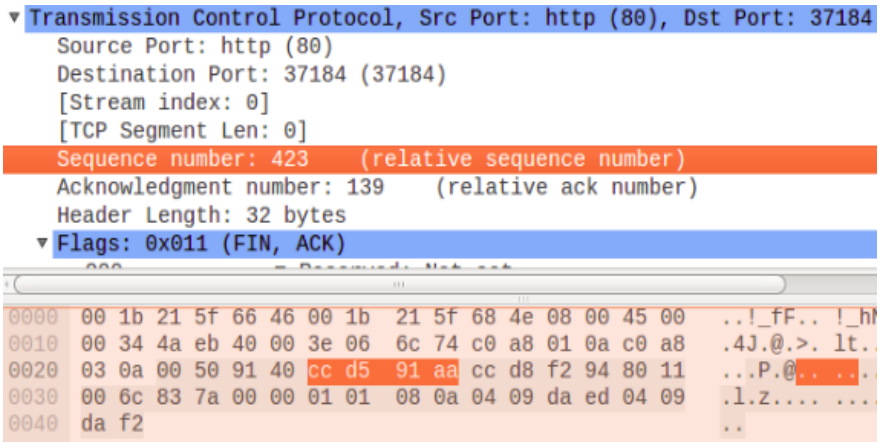
Der spezifizierte Port, von welchem der Sender sendet.

◇ Empfänger TCP Port Nummer (2 Byte)

Der spezifizierte Port, wo der Empfänger die Nachricht erhält.

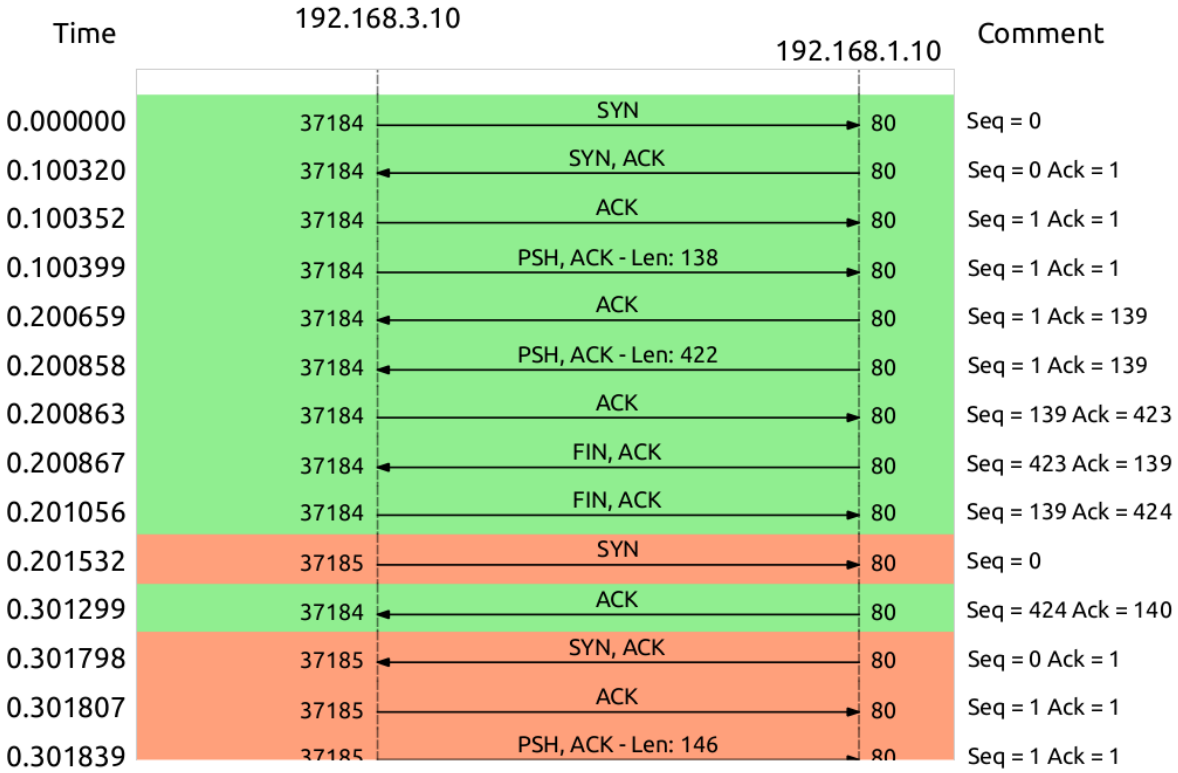
◇ Sequenznummer (4 Byte)

... sie dient der Information über die Menge bereits gesendeter Data. Sie ist in jedem Datenpacket vorhanden und dient mitsamt der Acknowledge Nummer (next §) dazu, dem Sender den erfolgreichen Transfer zu übermitteln. Zwecks Sicherheitsvorkehrungen startet diese immer mit einem zufälligen Wert zwischen 0 und $(2^32) - 1$. Wireshark bietet hinsichtlich der X-hunderten Zahlenwerten eine relative Sequenznummer an, um die Packetidentifikation in mehreren Hinsichten zu vereinfachen.



◇ Acknowledge number (4 Byte)

Diese dient dazu den Datentransfer zu bestätigen. Prinzipiell gilt, dass die geantwortete ACK zur neuen SEQ wird, siehe auch



... Via Wireshark, lässt sich ein solcher Graph mittels

`Statistics > Flow Graph > Flow Type: TCP flow`

erstellen.

◇ TCP Data Offset (X words ... 8 Bytes each)

.. dieser dient zur Spezifikation des eigentlichen Dateistarts. Anhand von 'Header Length: xx Byte' lässt sich dies feststellen.

◇ Reserved Data (4 Bit)

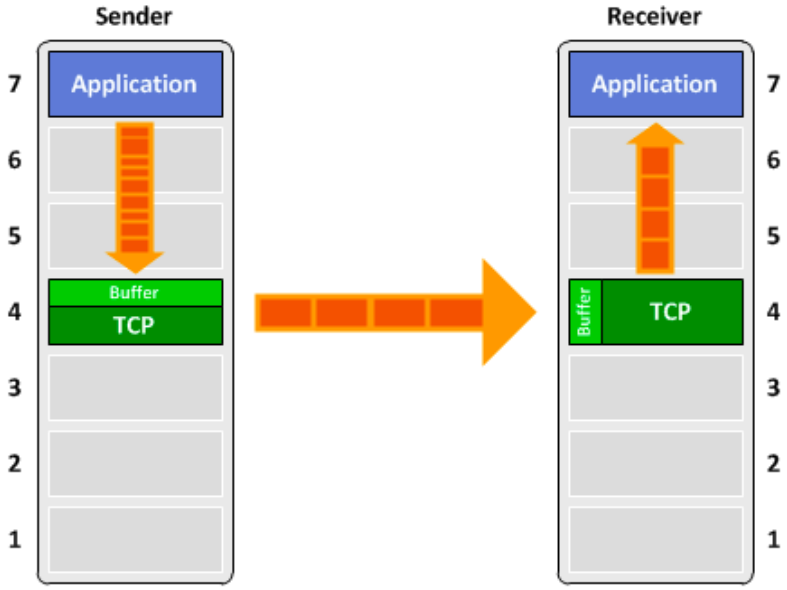
Das Reserved Data-Feld ist für zukünftige Verwendungen reserviert. Alle Bits müssen null sein.

◇ Control Flags (8 Bit)

Beschrieben durch 2 Variablen oder 4 Hexa-Werten. Zu den möglichen Flags gehören:

- ECE - Explicit Congestion Notification (ECN-ECHO): Bei Netzwerküberlastung
- CWR - Congestion Window Reduced: Revertiert ECE
- URG - Urgent: Selten genutzt, oftmals als Interrupt-Signal
- ACK - Acknowledgement: Setzt Gültigkeit für Sequenz/Acknowledge-Values
- PSH - Push: Umgehung lokaler Puffer für schnellere Übertragung
- RST - Reset: Zum Abbruch von Verbindungen
- SYN - Initiieren der Verbindung. Üblicherweise beantwortet mit SYN+ACK oder RST
- FIN - Finish: Schlussflag, dient zur Freigabe der Verbindung & Bestätigt vollständige Übertragung.

Bezüglich PSH: Datenverkehr & Pufferfunktion bei Übertragung



◇ Window size (2 Bytes)

Bestimmt die Puffergröße bezüglich der zu sendenden Daten. Dies steht in Relation mit ACK, da der Datensatz erst mit der ACK-Nummer bestätigt wird. e.g.: Bei zu klein gewähltem Puffer muss der Sender eines Elementes auf die Antwort des Empfängers warten, bevor dieser weitere Packete des jeweiligen Elementes senden darf.

◇ TCP checksum[1] (2 Byte)

Die Prüfsumme des TCP-Headers dient der Erkennung von Fehlübertragungen, bestehend aus Empfänger-IP, Sender-IP, TCP-Protokollerkennung, Länge des TCP-Headers und Nutzdaten.

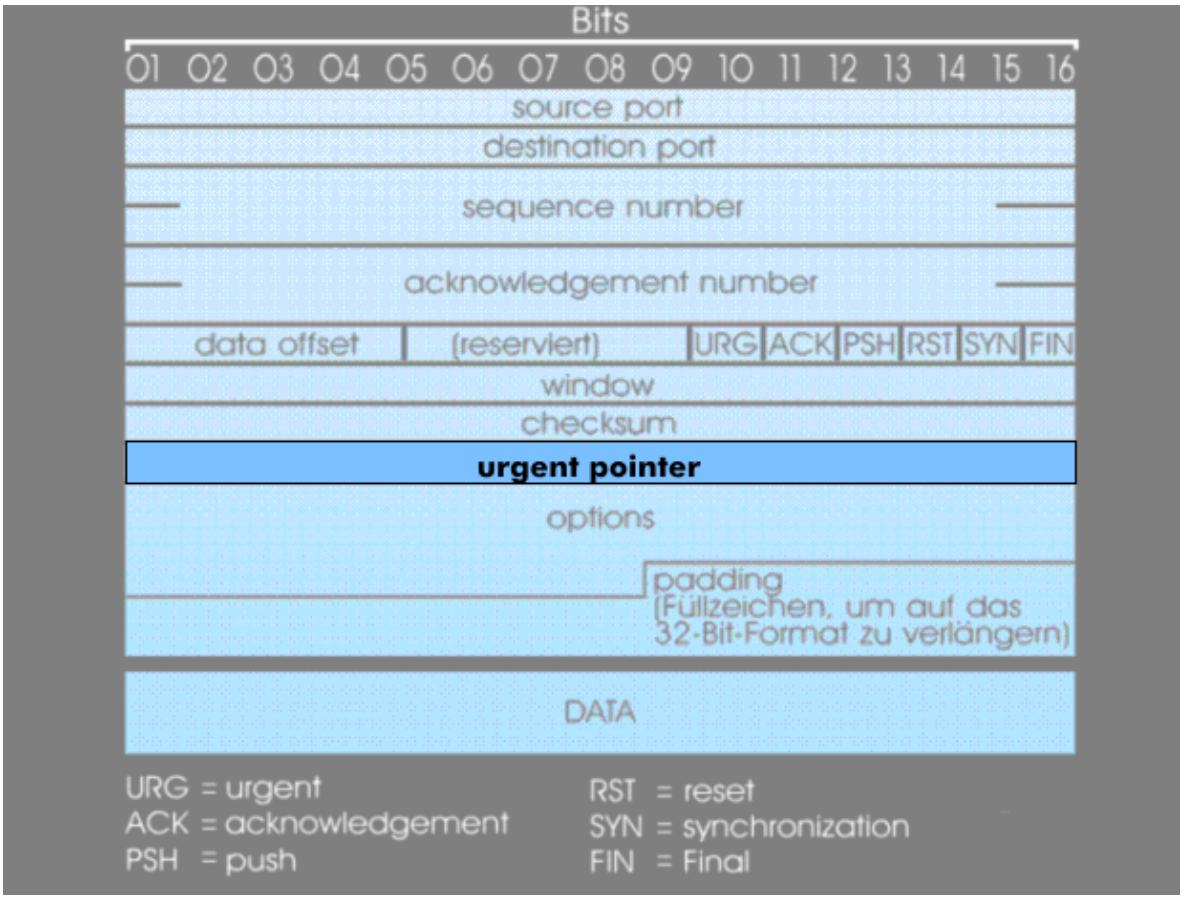
◇ Urgent Pointer (2 Byte)

Der Urgent-Pointer zeigt auf das Ende der dringenden Daten. Der Wert des Pointers ist ein positiver Offset der Sequenznummer. Gilt nur wenn die URG-Flag gesetzt ist.

◇ Options (0 .. 40 Byte)

Das Options-Feld ist immer unterschiedlich groß, je nach Zusatzinformationen. Eine Option muss stets die RFC-Bedingung von 8 Bit erfüllen, dementsprechend muss man potenzielle Optionswerte mit Nullwerten padden. Man kann beispielsweise die Maximum Segment Size, MSS, beeinflussen. Per Default ist dieser: IP Datagram mit 576 Byte - 40 Byte TCP Header = 536 Byte.

[1] http://www.roman10.net/2011/11/27/how-to-calculate-iptcpudp-checksumpart-1-theory/



Dementsprechen lassen sich dann die Antworten und Nachfragen des Servers interpretieren.

1.3 Wie viele TCP-Verbindungen werden insgesamt aufgebaut? Wie unterscheidet sich das von dump_protocols.pcap?
Mithilfe von

```
Analyze > Follow > TCP Stream
```

können wir durch die verschiedenen TCP-Verbindungen tooglen. Im Rahmen dessen können wir auch einsehen, was genau übertragen wurde.
Für unser .pcap stehen 3 unterschiedliche TCP-Verbindungen.

- Stream 0: Beschreibt GET /test/
- Stream 1: Beschreibt GET /test/logo.gif
- Stream 2: Beschreibt GET /test/TechnikErleben.png

Es handelt sich dabei um eine altbewährte Form der HTTP/1.0 Datenübertragung - 1 eigene TCP-Verbindung per Grafik.

1.4 Bestimmen Sie, wie viele Bytes in jeder Verbindung ausgetauscht werden und wie lange die einzelnen Verbindungen bestehen. Via

```
Statistics > Conversations ... TCP
```

können wir alles rund um die TCP-Verbindungen feststellen.