Christian Winkler

chris@bashit.me

Abstract

Dieses Dokument beschreibt den Aufbau, potentielle Sicherheitslücken  
und die Funktionen und Features der Protokolle TCP und IPv4.

Dokumentation

TCP/IPv4

Table of Contents

[TCP/IPv4 3](#_Toc527379726)

[TCP 3](#_Toc527379727)

[Verbindungsaufbau 3](#_Toc527379728)

[Verbindungsabbau 3](#_Toc527379729)

[Weitere Flags 3](#_Toc527379730)

[Buffer 3](#_Toc527379731)

[Datenintegrität 3](#_Toc527379732)

[Header 4](#_Toc527379733)

[Timer 5](#_Toc527379734)

[Flow Control 6](#_Toc527379735)

[Congestion Control 7](#_Toc527379736)

[IPv4 9](#_Toc527379737)

[Einsatzbereiche 9](#_Toc527379738)

[Adressformat 9](#_Toc527379739)

[Header 9](#_Toc527379740)

[Routing 10](#_Toc527379741)

[Netzklassen (Nicht mehr in Gebrauch) 10](#_Toc527379742)

[Besondere Adressen 11](#_Toc527379743)

[ICMP 11](#_Toc527379744)

[Höhere Protokolle 11](#_Toc527379745)

[Datagramm 12](#_Toc527379746)

[Datagrammfragmentierung 12](#_Toc527379747)

[Sicherheitslücken 13](#_Toc527379748)

[(IP) Source Routing 13](#_Toc527379749)

[TCP Sequence Prediction 13](#_Toc527379750)

[TCP Blind Spoofing 13](#_Toc527379751)

[SYN Flooding 13](#_Toc527379752)

[Man-in-the-middle Attacke 13](#_Toc527379753)

[DNS Protokoll Attacke 14](#_Toc527379754)

[Application Layer Attacks 14](#_Toc527379755)

[Flooding Attacken 14](#_Toc527379756)

[Rogue Devices 14](#_Toc527379757)

# Table of Figures

[Abbildung 1: TCP Header (Source Wikipedia) 3](#_Toc527379485)

[Abbildung 2: TCP Receive Buffer (Source Brainstorti) 4](#_Toc527379486)

[Abbildung 3: Slow-Start und Congestion Avoidance Algorithmus (Source Wikimedia) 4](#_Toc527379487)

# List of Tables

[Tabelle 1: IPv4 Header (Source Wikipedia) 6](#_Toc527379602)

[Tabelle 2: Besondere Adressen (Source RFC6890 - IETF) 8](#_Toc527379603)

# TCP/IPv4

## TCP

Das Transmission Control Protocol definiert auf welche Art Daten zwischen Netzwerkkomponenten ausgetauscht werden. Eine TCP Verbindung ist im Prinzip eine Ende-zu-Ende-Verbindung in Vollduplex, kann aber auch als 2 Halbduplexverbindungen (bei denen Informationen nicht gleichzeitig in beide Richtungen fließen können) betrachtet werden. Eine TCP Verbindung wird durch 2 Endpunkte identifiziert. (Lokale IP, Lokaler Port, Entfernte IP, Entfernter Port) Ein solches Paar nennt man Socket.

### Verbindungsaufbau

Client -> <SEQ=100><CTL=SYN> -> Server // <> - Flag im Header  
Client <- <SEQ=300><ACK=101><CTL=SYN,ACK> <- Server  
Client -> <SEQ=101><ACK=301><CTL=ACK> -> Server

(seq -> ack [seq+1] | seq = server sequence | syn = synchronize | ack = acknowledgement)

### Verbindungsabbau

Client -> <SEQ=205><CTL=FIN> -> Server  
Client <- <SEQ=404><ACK=206><CTL=ACK> <- Server  
Client <- <SEQ=404><CTL=FIN> <- Server  
Client -> <SEQ=206><ACK=405><CTL=ACK> -> Server

### Weitere Flags

CWR (Congestion Window Reduced) und ECE (Explicit Congestion Notification Echo) werden ausgetauscht, wenn das Netzwerk des Empfängers überlastet ist (Client sendet ECE, Server CWR wenn die Übertragungsrate reduziert wurde).  
URG = urgent  
PSH = push (Deaktiviert Puffer – Wichtig bei schnellen Anwendungen, telnet, Terminals usw.)  
RST = reset (Wenn die Verbindung bei z.B. Problemen Abgebrochen werden muss)

### Buffer

Beim Übertragen von Daten werden 2 Buffer verwendet. Der auf der Senderseite wartet auf mehr Daten, um sie effizienter in einem Paket zu senden. Der auf der Empfängerseite wartet auf mehrere Pakete um sie gebündelt an die Applikation weiterzugeben.

### Datenintegrität

Da Daten oft doppelt versendet werden oder verloren gehen können sorgt das TCP dafür, dass die Fehler der anderen Schichten ausgebessert werden. Die Integrität der Daten wird mittels Prüfsumme im Paketkopf überprüft und die Reihenfolge der Daten wird durch Sequenznummern sichergestellt.

### Header

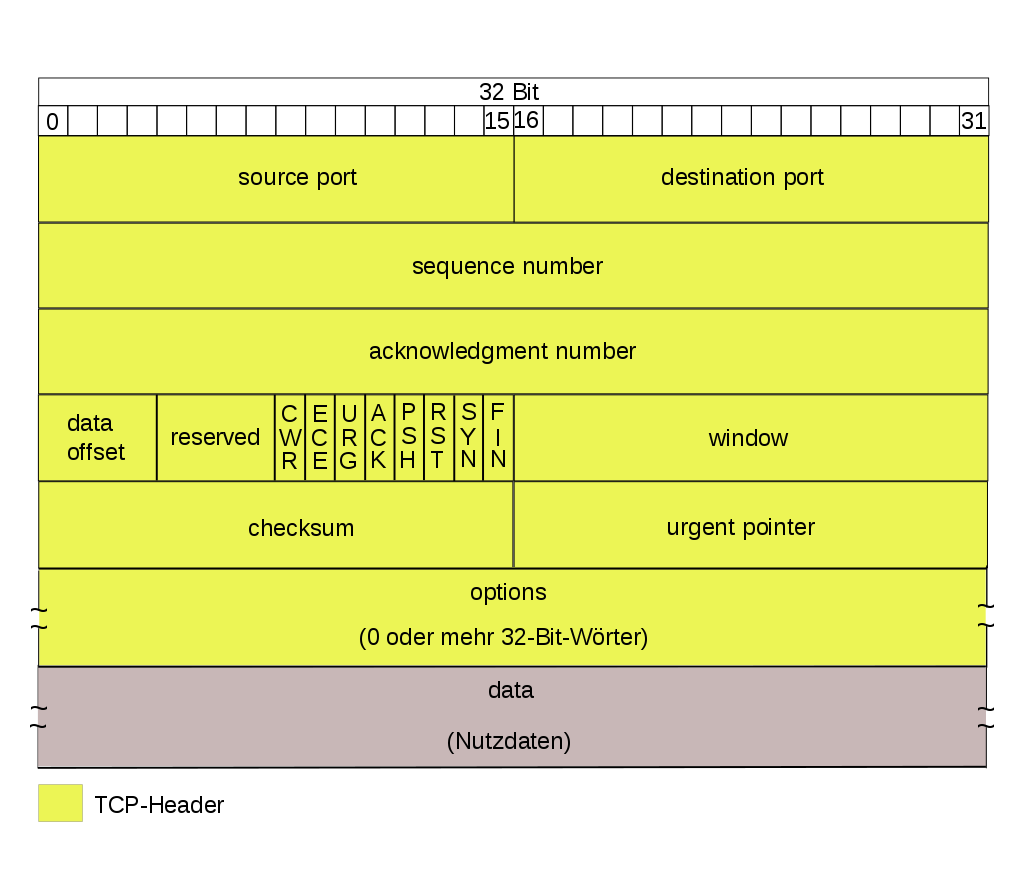


Abbildung 1: TCP Header (Source Wikipedia)

#### Acknowledgement Number

Gibt die Sequenznummer an, die der Absender als nächstes erwartet. Nur gültig, wenn das ACK Flag gesetzt ist.

#### Data Offset

Länge des TCP Headers (Startadresse der Nutzdaten).

#### Reserved

Reserviert für zukünftige Anwendungen.

#### Window

Anzahl der Bytes, beginnend bei dem durch ACK gekennzeichneten Byte, die der Sender des TCP Paketes bereit ist zu empfangen.

#### Urgent Pointer

Gibt die Position des ersten Bytes nach den „urgent“ Daten an. Nur gültig, wenn das URG Flag gesetzt ist.

### Timer

#### Retransmission Timer

Um verlorene Segmente erneut zu übertragen, verwendet TCP das Retransmission Timeout (RTO). Wenn TCP ein Segment sendet, startet der Timer. Wenn die Bestätigung empfangen wird, stoppt der Timer wieder. Wenn der Timer abläuft, tritt ein Timeout auf und das Segment wird erneut übertragen. RTO gilt für 1 RTT (Round Trip Time). Zur Berechnung des RTO muss zuerst die RTT berechnet werden. Die meisten Implementierungen verwenden folgende Formel zur Berechnung des RTO:

“ RTO = RTTs + 4\* RTTd “

##### Measured RTT (RTTm)

Die RTTm ist die Zeit für das Senden des Pakets und das Empfangen der Bestätigung.

##### Smoothed RTT (RTTs)

Der gewichtete Durchschnitt der RTTm. Die Fluktuation der RTTm ist zu hoch um für die Berechnung des RTO herangezogen zu werden.

##### Deviated RTT (RTTd)

Die meisten Implementierungen verwenden nicht nur RTTs zur Berechnung des RTO, sondern auch die RTTd. RTTd ist die Standardabweichung der RTTm.

#### Persistent Timer

#### Zum behandlen von einem Spezialfall. Wenn das Receive Window 0 ist und die Nachricht zum weitersenden verloren geht wartet der Server endlos. Wenn der Sender eine RW größe von 0 erhält startet er den Persistenztimer. Wenn der Persistenztimer abläuft, sendet der Sender ein spezielles Segment, das als Sonde bezeichnet wird. Dieses Segment enthält nur 1 Byte neue Daten. Es hat eine Sequenznummer, aber die Sequenznummer wird nie bestätigt. Sie wird sogar bei der Berechnung der Sequenznummern für den Rest der Daten ignoriert. Die Sonde veranlasst den Empfänger, die verloren gegangene Bestätigung erneut zu senden.

#### Quiet Time

Der Timer startet nach dem Senden des letzten ACK für das 2. FIN und dem Schließen der Verbindung. Nach dem Schließen einer TCP-Verbindung ist es möglich, dass Datagramme, die sich noch auf dem Weg durch das Netzwerk befinden, versuchen, auf den geschlossenen Port zuzugreifen. Der Quiet Time Timer soll verhindern, dass sich der gerade geschlossene Port schnell wieder öffnet und diese letzten Datagramme empfängt.

#### Keepalive-Timer

Ein Keepalive-Timer wird verwendet, um eine lange Leerlaufverbindung zwischen zwei TCPs zu verhindern. Wenn ein Client eine TCP-Verbindung zu einem Server öffnet, überträgt er einige Daten und wird still. In diesem Fall bleibt die Verbindung für immer offen. Daher wird ein Keepalive-Timer verwendet. Jedes Mal, wenn der Server von einem Client hört, setzt er diesen Timer zurück. Die Auszeit beträgt in der Regel 2 Stunden. Wenn der Server nach 2 Stunden nichts vom Client hört, sendet er ein leeres Datenpaket mit aktiven ACK Flag. Wenn nach 10 Paketen, von denen jedes 75 Sekunden auseinander liegt, keine Antwort erfolgt, geht der Server davon aus, dass der Client down ist und beendet die Verbindung.

### Flow Control

Da TCP einen Buffer auf der empfangenden Seite nutzt und Applikationen laufend aus dem Buffer lesen muss es eine Flusssteuerung geben. Der Empfänger legt ein sogenanntes „Receive Window“ fest, dass kleiner als die Buffergröße ist.

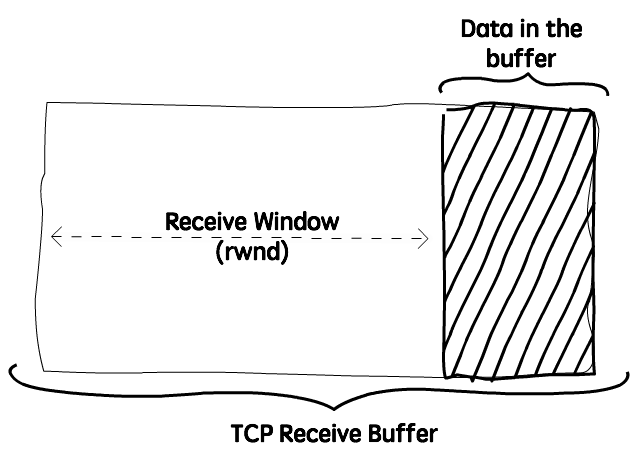


Abbildung 2: TCP Receive Buffer (Source Brainstorti)

Um die Flow Control leichter zu erklären betrachten wir folgendes Beispiel. Wir wollen eine Datei mit 150000 Byte übertragen. TCP könnte diese Datei in 100 Pakete mit jeweils 1500 Bytes aufteilen. Der Empfänger hat zufälligerweise ein Receive Window von 45000 Bytes, dass uns die Berechnung der Datenpakete, die ohne ACK zugleich übertragen werden können, einfach macht. TCP sieht, dass 30 Pakete à 1500 Bytes (45000 Bytes) in das Receive Window passen. TCP sendet die ersten 30 Pakete ab und erhält z.B. für die ersten 10 ein ACK. Jetzt werden nur noch 20 Pakete übertragen und TCP kann die nächsten 10 senden. Diese Grenze von 30 Paketen nennt man „Sliding Window“.

### Congestion Control

#### Slow Start and Congestion Avoidance

Zu Beginn einer Datenübertragung wird der Slow Start and Congestion Avoidance Algorithmus genutzt um zu bestimmen, wie viele Daten gesendet werden können, bis das Netz überlastet ist. Dies wird gemacht um Überlastsituationen vorzubeugen.

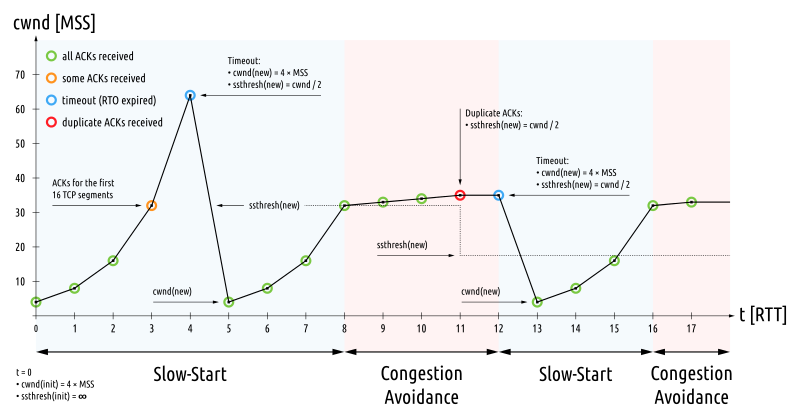


Abbildung 3: Slow-Start und Congestion Avoidance Algorithmus (Source Wikimedia)

##### Slow Start

Das congestion window beschreibt, wie Viele Pakete in einer RTT an den Empfänger gesendet werden können. Für jede RTT wird diese Anzahl verdoppelt, bis es zu einem Timeout kommt. Der sogenannte Slow-Start Threshold wird auf die MMS (Maximum Segment Size) gesetzt, bei der zuletzt ACK Pakete empfangen wurden. Nachdem dies geschieht wechselt der Algorithmus in die Congestion Avoidance Phase.

##### Congestion Avoidance

In dieser Phase wird die MMS nur noch um 1 pro RTT erhöht bis der Slow-Start Threshold erreicht wird.

#### Fast-Retransmit and Fast-Recovery

Hierbei bestätigt der Empfänger nach einem Paketverlust für jedes eintreffende Paket das Paket nach dem, das verloren wurde, um dem Sender schnell mitzuteilen, dass ein Paket verloren ging. Dadurch muss das Timeout nicht abgewartet werden. Nach dem dritten „duplicate ACK“ sendet der Sender das verlorene Paket erneut. Nach einem Fehler wird hierbei das congestion window nur halbiert anstatt wieder mit einem Slow Start zu beginnen.

#### SACK (Selective ACKs)

Wird verwendet um noch mehr Informationen über den Datenfluss zu erhalten. Nach einem Paketverlust sendet der Empfänger dem Sender ein TCP Paket, in dessen Header genau steht, welche Pakete bereits angekommen sind und welche noch fehlen.

#### TCP-Tahoe

Nach einem Fehler wird das congestion window auf 1 gesetzt und der Slow-Start Algorithmus beginnt von neuem.

#### TCP-Reno

Nach einem Fehler wird zwischen Timeout und duplicate ACKs unterschieden. Somit wird das congestion window nach mehreren duplicate ACKs nicht auf 1 gesetzt, sondern halbiert. (Siehe Fast-Retransmit and Fast-Recovery)

## IPv4

IPv4 ist die vierte Version des Internet Protokolls und auch die erste Version, die Weltweit verbreitet wurde. (\*weiter IP)

### Einsatzbereiche

IPv4 wird nicht nur im Ethernet so wie wir es kennen eingesetzt, sondern kann auf verschiedenste Medien wie z.B. serielle Schnittstellen (PPP oder SLIP) oder Satellitenverbindungen aufsitzen.

### Adressformat

IPv4 benutzt 32-Bit Adressen, die üblicherweise dezimal in 4 Blöcken getrennt mit Punkten und ohne vorangestellten nullen geschrieben wird. Jeder Block repräsentiert 8 Bit womit sich ein Wertebereich von 0-255 für jeden Block ergibt. Eine IP-Adresse besteht aus einem Netzanteil und einem Hostanteil. Der Netzanteil identifiziert ein Teilnetz, der Hostanteil identifiziert ein Gerät innerhalb eines Teilnetzes. Die Aufteilung zwischen Netzanteil und Hostanteil wird durch eine Subnetzmaske festgelegt. Diese wird entweder auch wie eine IP Adresse mit Blöcken von entweder 255 oder 0 angegeben oder in der CIDR-Notation. Die CIDR-Notation gibt an wie viele Bits der Subnetzmaske am Anfang 1 sind. 255.255.255.0 -> 11111111.11111111.11111111.00000000 -> 24 -> Schreibweise IP/24

### Header

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0–3** | **4–7** | **8–11** | **12–15** | **16–18** | **19–23** | **24–27** | **28–31** |
| Version | IHL | Type of Service | | Gesamtlänge | | | |
| Identifikation | | | | Flags | Fragment Offset | | |
| TTL | | Protokoll | | Header-Prüfsumme | | | |
| Quell-IP-Adresse | | | | | | | |
| Ziel-IP-Adresse | | | | | | | |
| *Optionen …* | | | | | | | |

Tabelle 1: IPv4 Header (Source Wikipedia)

#### IHL (Internet Header Length)

Gibt die Länge des IP-Headers in 32 Bit an. (15 = 15x32 Bit)

#### Type of Service

Diente ursprünglich als Entscheidungshilfe für beteiligte Router an der Wahl der Übertragungsparameter. Heute wird dieses Feld meist zur Vermeidung von Überlastungen verwendet.

#### Gesamtlänge

Gesamtlänge des IP-Pakets. Abzüglich der IHL ergibt es die Größe der Nutzdaten.

#### Identifikation

Wird zur Nummerierung der Datenpakete verwendet und ist eindeutig und fortlaufend.

#### Flags

Die drei Flags werden zur Datenfragmentierung genutzt. Das erste Flag ist immer 0, das 2. Flag (DF) gibt an, dass das Datenpaket nicht fragmentiert werden darf und das 3. Flag (MF) gibt weitere Fragmente an, wenn es gesetzt ist.

#### Fragment Offset

Enthält die Position des Datenfragments im ursprünglichen IP-Paket an.

#### TTL (Time-To-Live)

Mit der TTL gibt der Sender die Lebensdauer des Pakets an. Jeder Punkt im Netzwerk, der das Paket weiterleitet zieht von der TTL mindestens 1 ab und verwirft es, wenn die TTL 0 erreicht. Damit wird verhindert, dass Pakete ewig leben, wenn sie nicht zustellbar sind.

#### Protokoll

Enthält den Port des übergeordneten Transport-Protokolls z.B. TCP oder UDP.

### Routing

Beim Routing über IP können Pakete verlorengehen, doppelt übertragen werden, verschiedene Wege nehmen und fragmentiert beim Empfänger ankommen. Wird TCP auf IP aufgesetzt, wird der Paketverlust durch Wiederholung korrigiert und doppelte Pakete werden verworfen. Die Kombination aus TCP und IP stellt dabei eine zuverlässige bidirektionale Verbindung eines Datenstroms dar.

### Netzklassen (Nicht mehr in Gebrauch)

Von 1981-1993 wurde IPv4 in verschiedenste Netzklassen geteilt. Von der Netzklasse kann man die Größe des Netzes ablesen (es gab keine Subnetzmaske). Dies war damals wichtig um beim Routing fremde Netze vom eigenen zu unterscheiden.

#### Class A

Netze 0.0.0.0 bis 127.255.255.255 (24-Bit Host - /8)

#### Class B

Netze 128.0.0.0 bis 191.255.255.255 (16-Bit Host - /16)

#### Class C

Netze 192.0.0.0 bis 223.255.255.255 (8-Bit Host - /24)

#### Class D

Multicast-Gruppen 224.0.0.0 bis 239.255.255.255 (28-Bit Multicast-Gruppen-ID - /4)

#### Class E

Reserviert 240.0.0.0 bis 255.255.255.255 (28-Bit für zukünftige Anwendungen - /4)

### Besondere Adressen

Einige Adressbereiche sind für spezielle Zwecke reserviert. Diese Reservierungen wurden von der Internet Engineering Task Force, von der auch IPv4 stammt, entwickelt und im „Request for Comments“ 6890 festgelegt. (Wikipedia Tabelle Falsch -> lieber im RFC6890 nachschauen)

|  |  |
| --- | --- |
| IP Range | Beschreibung |
| 0.0.0.0/8 | Der Host befindet sich in diesem Netz. |
| 10.0.0.0/8 | privates 8-Bit-Netzwerk |
| 100.64.0.0/10 | Shared Address Space |
| 127.0.0.0/8 | Loopback (Lokaler Computer) |
| 169.254.0.0/16 | Link Local |
| 172.16.0.0/12 | private 16-Bit-Netzwerke |
| 192.0.0.0/24 | IETF Protocol Assignments |
| 192.0.0.0/29 | DS-Lite |
| 192.0.2.0/24 | Documentation (TEST-NET-1) |
| 192.88.99.0/24 | 6to4 Relay Anycast |
| 192.168.0.0/16 | private 24-Bit-Netzwerke |
| 198.18.0.0/15 | Benchmarking |
| 198.51.100.0/24 | Documentation (TEST-NET-2) |
| 203.0.113.0/24 | Documentation (TEST-NET-3) |
| 240.0.0.0/4 | Reserviert |
| 255.255.255.255/32 | Limited Broadcast |

Tabelle 2: Besondere Adressen (Source RFC6890 - IETF)

### ICMP

Das Internet Control Message Protocol ist sozusagen eine Zusatzfunktion zum Internet Protocol (bei IPv6 heißt es ICMPv6). Es setzt auf IPv4 auf, befindet sich aber im TCP/IP-Protokollstapel immer noch in der Schicht Internet. Es dient dem Austausch von Informations- und Fehlermeldungen im IP. Da es Bestandteil vom IP ist wird von jedem Gerät erwartet, dass es ICMP versteht.

### Höhere Protokolle

Auf IPv4 setzen mehrere Protokolle auf. Direkt wären das TCP und UDP. UDP behält im Gegensatz zu TCP die Paketeigenschaften von IPv4 im Prinzip bei. Es ist verbindungslos, unzuverlässig, erlaubt doppelte Pakete usw… Auf TCP und UDP setzen dann weitere Protokolle wie DNS, HTTP, IMAP oder SMTP auf.

### Datagramm

Ein Datagramm ist eine in sich geschlossene, unabhängige Dateneinheit, die ohne weitere Verbindungssicherung zwischen zwei Endpunkten verschickt wird. Es enthält im Wesentlichen die Empfangs-, Absenderadresse und die Nutzdaten. Nachteile eines Datagrammes sind, dass volle Adressen mitgesendet werden müssen, dass nicht überprüft wird, ob der Empfänger überhaupt existiert und es gibt in einem Datagrammnetz keine Garantie dafür, dass die Daten in richtiger Reihenfolge ankommen.

### Datagrammfragmentierung

Wenn ein Datagramm an einem Punkt im Netzwerk zu groß für das nächste Teilnetz ist muss es aufgeteilt werden. Dazu sind folgende Schritte notwendig:

* Aufteilen der Nutzdaten an einer 64-Bit-Grenze (das zweite Fragment enthält dann nicht unbedingt ein Vielfaches von 64 Bit Daten)
* Kopieren der Headerdaten des Originaldatagramms in die neuen Header
* Setzen des „more-fragments“-Flags beim ersten Fragment
* Beim zweiten Fragment erhält das more-fragments Flag den Wert des Originaldatagramms, da das Originaldatagramm bereits ein Fragment gewesen sein kann.
* Erneutes Setzen der Länge-Felder in den Headern
* Beim zweiten Fragment enthält Fragment-Offset die Summe aus Fragment-Offset des Originaldatagramms und Anzahl (Nutzdaten-)Bytes im ersten Fragment.

## Sicherheitslücken

### (IP) Source Routing

Bei dieser Art von Attacke greift der Angreifer in die Datenfragmention ein. Wenn das Internet Protokoll Daten fragmentiert legt es eine genaue Route für die Datenpakete fest. Hierbei kann der Angreifer die Route ändern.

### TCP Sequence Prediction

Da TCP Pakete eine Sequenz generieren kann ein Angreifer über Algorithmen die nächste Sequenz voraussagen und damit Attacken wie TCP Blind Spoofing ausführen.

### TCP Blind Spoofing

Wenn der Angreifer den Port und die nächste Sequenznummer weiß kann er in einen laufenden Datenstrom einsteigen und sich als Sender ausgeben.

### SYN Flooding

Ein Angreifer kann SYN Pakete mit falschen Source Adressen an einen Server schicken. Der Server antwortet dann mit SYN-ACK Paketen, auf die nie eine Antwort kommen wird. Dadurch entstehen am Server dann haufenweise halboffene Verbindungen und der Server antwortet irgendwann nicht mehr.

### Man-in-the-middle Attacke

#### Session Hijacking (by Sniffing)

Beim Sniffing könnte der Angreifer auf eine HTTP Session stoßen und damit Accounts auf Webseiten übernehmen und im Namen des Opfers handeln.

#### Replay Attacke

Beim Sniffing könnte der Angreifer auf eine Session ID oder gehashte Daten treffen. Der Angreifer sendet diese Daten dann erneut an den Webserver und könnte so Zugriff auf den Nutzeraccount erhalten.

#### Sniffing Attacke (MAC/ARP Overflow im Switch)

Bei einer Sniffing Attacke spielt der Angreifer dem Nutzer meist vor das Gateway zu sein. Er leitet also den Traffic des Opfers über seinen PC und kann somit die Daten ansehen. Eine alternative in Netzwerken mit Switch ist es vorher den Switch mit einer sogenannten ARP (Address Resolution Protocol) Overflow Attacke anzugreifen. Der Switch speichert MAC Adressen in einer ARP Tabelle. Wenn der Angreifer sich nun mit verschiedensten MAC Adressen am Switch meldet, bis dessen Tabelle volläuft wechselt der Switch in den Hub Modus. Im Hub Modus sendet der Switch Datenpakete nicht nur an den richtigen Empfänger sondern Broadcastet sie und der Angreifer kann dadurch leicht den Datenverkehr überwachen. Alle Attacken dieser Art nennt man Man in the middle Attacken.

### DNS Protokoll Attacke

Hierbei werden z.B. durch eine Fast-Flux DNS Attacke oder Cache Poisoning DNS Einträge ausgetauscht. Dadurch kann ein Angreifer ganze Webseiten vortäuschen.

### Application Layer Attacks

Buffer Overflow/Underflow Attacken, Viren und Trojaner.

### Flooding Attacken

Der Angreifer leitet so lange viel Traffic auf ein Netzwerkgerät, bis dieses Gerät nicht mehr reagiert. Hierbei unterscheidet man noch DoS und DDoS Attacken. Eine Denial of Service Attacke wird von einem Computer und über eine Verbindung ausgeführt. Eine Distributed Denial of Service Attacke wird von mehreren Computern (meistens ein Botnet) und über mehrere Verbindungen ausgeführt.

### Rogue Devices

Geräte, die eigentlich nicht Teil des Netzwerkes sein sollten.

#### Rogue access points

Router, die von einem Hacker oder Mitarbeiter an das Netzwerk angeschlossen werden. Über solche Geräte können weitere unautorisierte Geräte auf das Netzwerk zugreifen oder Daten gestohlen werden.

#### Rogue peers

Computer, die ohne Berechtigung mit dem Netzwerk verbunden sind.