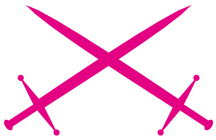

Firewalls umgehen mit Protokoll-Tunneling

Steffen Wendzel

Schwierigkeitsgrad

Beim Tunneling von Protokollen werden einzelne Protokolle, Daten oder Verbindungen innerhalb eines anderen Protokolls transportiert. Das Tunneling ist dabei für die Endanwendungen transparent. Dieser Artikel beschreibt, wie Tunneling mit verschiedensten Protokollen funktioniert und wie man es anwendet um Firewalls zu umgehen.

Tunneling ist eine in modernen Netzwerken sehr häufig angewandte Technik zur Übertragung von Protokollen innerhalb anderer Protokolle. Beispielsweise werden Daten über VPN-Verbindungen (die ebenfalls als Tunnel zu bezeichnen sind) oder über SSH-Tunnel gesendet um diese zu authentisieren oder zu verschlüsseln. Ebenfalls häufig verwendet werden Tunnel um Ipv6-Pakete durch Ipv4-Netzwerke zu tunneln, was mittels 6to4-Protokoll realisiert wird. Wenn man möchte, dann kann man aber genauso gut Ipv6 über UDP oder TCP oder gar über Ipv6 selbst tunneln. Das gilt auch für das Ipv4 Protkoll, dass man ebenfalls via Ipv4 und allerlei möglichen anderen Protokolle tunneln kann.

Um das Tunneling nun aber wirklich zu verstehen muss man sich zunächst das TCP/IP Schichtenmodell ins Gedächtnis rufen (siehe Abbildung 1), das einige Leser sicherlich bereits kennen werden. Es besteht aus ingesamt vier Schichten, die in jedem TCP/IP-fähigen System soweit implementiert sind, wie sie benötigt werden.

Der Application Layer überträgt das Protokoll des jeweiligen Programmes - beispielsweise das POP3-Protokoll, um E-Mails abzuholen. Diesen Layer kann man recht einfach für das Tunneling einsetzen, da man sich nicht um das Routing oder die Fehlerüberprüfung der gesendeten und empfangenen Daten kümmern muss.

Das Tunneling und die Schichten des TCP/IP-Modells funktionieren auf die gleiche Weise: Protokolle werden innerhalb anderer Protokolle übertragen. Beim Schichtenmodell wird der Header sowie die Daten des

In diesem Artikel erfahren Sie...

- alles über das Konzept des Tunnelings
- Interessantes über die Technik, Firewalls mit Hilfe von Tunneling umzugehen

Was Sie vorher wissen/ können sollten...

- TCP/IP Grundkenntnisse
- Grundkenntnisse über Firewalls
- Erfahrung mit Unix Netzwerken

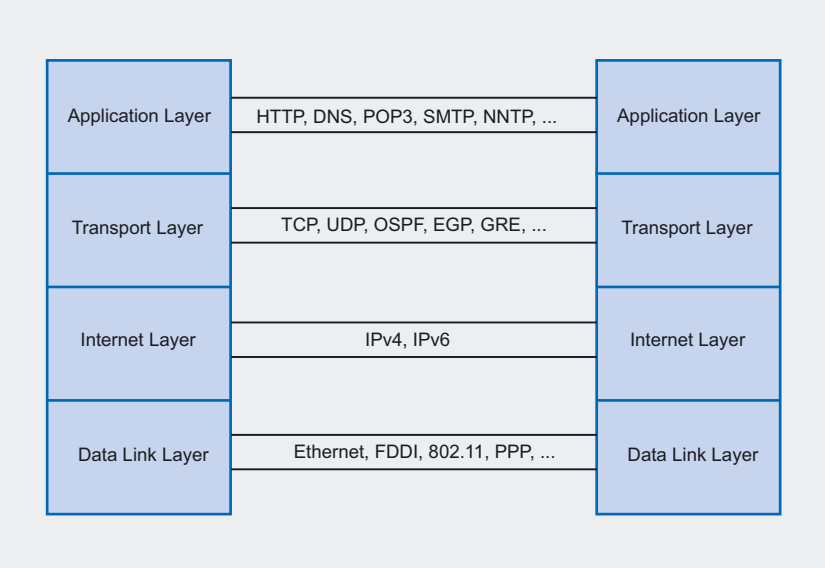

Abbildung 1. *Das Konzept des Tunnelings*

Application-Layers dabei an den Header des Protokolls im Tansport-Layer angehängt. Diese Daten wiederum werden an den Header des Internet-Layers angehängt. Zum Schluss werden die Daten des Internet-Layers noch in das Protokoll des Data Link-Layers eingebaut und versendet. Beim Empfang der Daten läuft alles umgekehrt. Dieses Verfahren kann man sich hervorragend an dem Modell Zwiebelschale merken, die man schichtenweise aufpellt.

Möchte man nun Daten tunneln, so bringt man diese Daten als Payload oder versteckt im Header eines anderen Protokolls unter. Dabei kann man diese Daten - verfügt man über genügend Kreativität – in fast jedem Protokoll tunneln und hervorragend verstecken.

IPSec-Tunnel

Ein recht einfach zu verstehendes Beispiel für einen Tunnel ist ein IPSec-Tunnel mit dem virtuelle private Netzwerke (VPNs) errichtet werden. Für das Tunneling kommen je nach Konfiguration dabei zwei Protokolle zum Einsatz: der *authentication header* (*AH*) und *encapsulated security payload* (*ESP*). Ersterer stellt die Authentizität der Daten sicher, letzterer kann für die Verschlüsselung und (allerdings in geringerem Maße als AH) ebenfalls für die Sicherstellung der Authentizität der Daten verwendet werden. IPSec kann so konfiguriert werden, dass es entweder im >transport mode< oder im >tunneling mode< arbeitet, wobei es die Daten dabei auf zwei verschiedenen Schichten tunnelt: Auf dem Internet- und auf dem Transport-Layer. (Ab. 2)

Im Transport-Mode wird der Header des IPSec Protokolls im IP-Header untergebracht. Der Header des Transport-Protokolls (z.B. TCP) und die dessen Payload enthaltenen Protokolle finden wiederum im IPSec-Header Platz und Schutz.

Im Tunnelig Mode wird hingegen ein komplettes IP-Datagram ersetzt. Zu diesem Zweck erzeugt IPSec einen neuen IP-Header, der den IPSec-Header enthält. Im IPSec-Header ist im Tunnelig Mode nicht mehr nur der Transport-Layer und Application-Layer, sondern auch der Internet-Layer geschützt.

Ein erster Tunnel

Genug Theorie fürs erste. Einen Tunnel mit fertiger Software einzurichten ist gar nicht so schwierig. Am einfachsten geht das mit dem auf praktisch jedem Linux-, Unix- und BSD-System installierten OpenSSH. Mit OpenSSH bekommt man nicht nur einen einfachen Tunnel, sondern gleichzeitig einen sehr sicheren und zudem verschlüsselten Tunnel zu Stande, ohne dass man sich um die Details des Protokolltunnelings kümmern muss (doch keine Sorge, gleich geht es weiter mit den Protokollen!).

Um einen einfachen SSH-Tunnel zu errichten benötigt man nicht einmal zwei Systeme: ein einziges auf dem ein TCP-Dienst läuft reicht völlig (auch wenn dies weniger Sinn macht). Doch egal, ob man die SSH-Verbindung für ein einziges oder zwei Systeme erstellt: Die Schritte, die man dazu durchführen muss, sind die Gleichen.

Im Folgenden soll eine IRC-Verbindung über SSH getunnelt werden. Der IRC-Server ist *server.xyz*. Wichtig ist, dass man auf diesem Rechner selbst über einen SSH-Zugang ver-

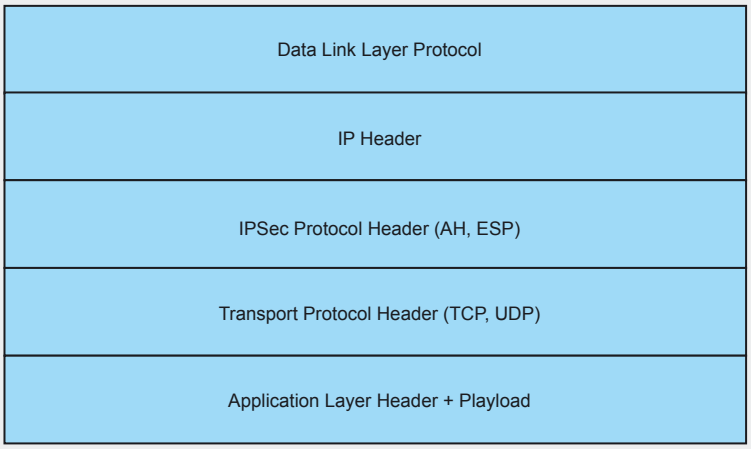

Abbildung 2. *Aufbau des IPSec-Headers im Transport-Mode*

Listing 1. *Eine Shell-Session via TCP-Streamsocket tunneln*

```
$ telnet target.xyz 80.
Trying 127.0.0.1...
Connected to target.xyz.
Escape character is '^]'.
ls;
...
: command not found
uname;
Linux
: command not found
uname -or;
2.6.15-26-386 GNU/Linux
: command not found
ps;
  PID TTY          TIME CMD
14716 pts/9    00:00:00 bash
14937 pts/9    00:00:00 sh
14955 pts/9    00:00:00 ps
: command not found
;;
sh: line 9: syntax error near
unexpected token `;;'
'h: line 9: `;;
Connection closed by
foreign host.
```

fügt, da die Verbindung über diesen Authentifiziert wird.

Dazu startet man SSH auf dem lokalen Rechner nach der folgenden Vorlage: Wobei gilt: *-f* bewirkt, dass SSH in den Hintergrund gestellt wird, *-C* bewirkt eine Kompression der Verbindung, *-N* bewirkt, dass auf der Gegenseite kein Befehl ausgeführt wird und mit *-L* gibt man den lokalen Port, danach den Hostname des Servers und dann den dortigen Port, jeweils durch Doppelpunkte sepparіert, an. Zum Schluss noch wie bei jeder SSH-Verbindung den Benutzernamen und den Host angeben.

```
$ ssh -f -C -N -L \
10000:server.xyz:6667 \
username@server.xyz
```

Dieser Aufruf würde also die IRC-Verbindung über den lokalen Port 10000 annehmen und auf *server.xyz* an Port 6667 weiterleiten. Jetzt kann man sich mit einem IRC-Client, etwa `irssi`, lokal auf Port 10000 verbinden.

```
$ irssi -c 127.0.0.1 -p 10000
```

Beachten Sie, dass Sie als nichtpriviligierter Nutzer unter Unix nur Ports ab 1024 und Aufwärts öffnen können. Was SSH dabei tut, dürfte mittlerweile klar sein: Die Verbindung unseres IRC-Clients wird innerhalb vom SSH-Protokoll getunnelt. Die Daten werden auf Port 10000 entgegengenommen und über das SSH-Protokoll verschlüsselt an den Zielrechner geschickt. Dieser empfängt und entschlüsselt diese Daten und sendet sie wiederum an die lokale Adresse auf Port 6667.

Genau das ist das Prinzip von Application-Layer basierten Tunneln, die zugleich Application-Layer Daten übertragen sollen: die eigentliche Verbindung wird über eine neue Verbindung geleitet, zu der sich der Client verbindet und von der aus Daten auch an den Server weitergeleitet werden. Für den Client läuft der Vorgang transparent ab. Er muss sich nur auf den lokalen, statt auf den entfernten Rechner verbinden.

Ganz einfach kann man sich übrigens auch mit *netcat* den Traum vom Tunnel wahr werden lassen und eine Shell-Verbindung (allerdings unverschlüsselt und ohne Passwortschutz) über Port 80 schicken. Diese Technik ist eine absolute Notlösung, falls keine anderen Programme zur Verfügung stehen. Auf dem Server-Rechner wird hierfür nc im listen-Modus gestartet (-l Paremter) und die Shell via `-e /bin/sh` ausgeführt: `nc -l -p 80 -e /bin/sh`.

Auf dem Client-Rechner hingegen muss man nur noch via Telnet eine Verbindung zum angebenen Port der Zielmaschiene aufbauen. Achten Sie darauf hinter jeden Befehl ein Semikolon zu schreiben, sonst funktioniert die Ausführung nicht. Indem man einen Fehler produziert (=zwei mal Semikolon eingeben), wird die Verbindung gekillt.(siehe Listing 1.)

Firewalls umgehen

Vielleicht kam dem Einen oder Anderen Leser nun schon die Idee, wie man denn eine Firewall mittels eines Tunnels umgehen kann. Um das zu erläutern stellen wir uns zunächst das übliche Szenario einer Firewall vor, die das interne Netzwerk eines Unternehmens mit dem Internet (oder einem beliebigen anderen Netzwerk) verbindet: Wenn die Firewall halbwegs vernünftig vom Administrator abgesichert wurde, dann lässt diese nur die Verbindungen durch, die ausserhalb auf die Ports zugreifen, die die Mitarbeiter benötigen. Im Normalfall handelt es sich dabei mit großer Sicherheit um die

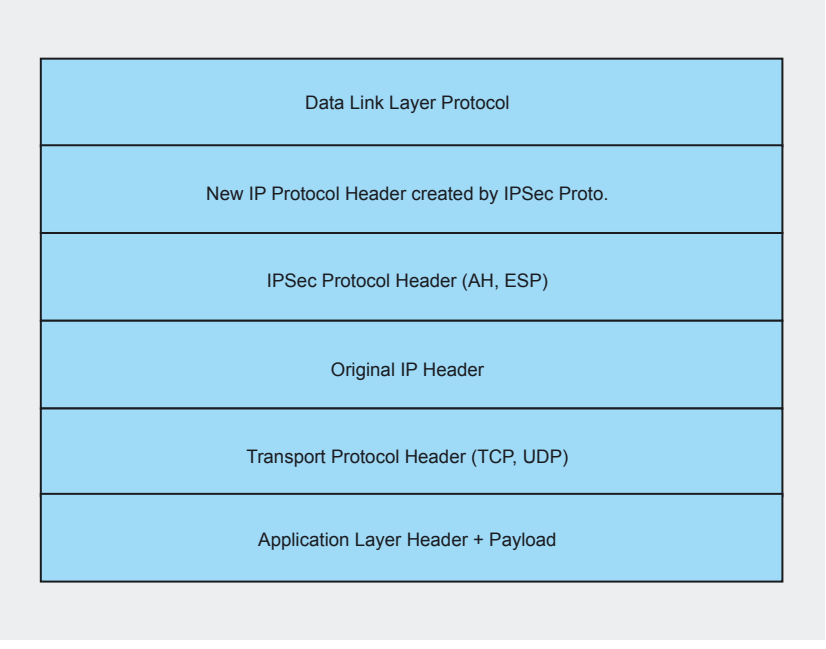

Abbildung 3. *Aufbau des IPSec-Headers im Tunneling-Mode*

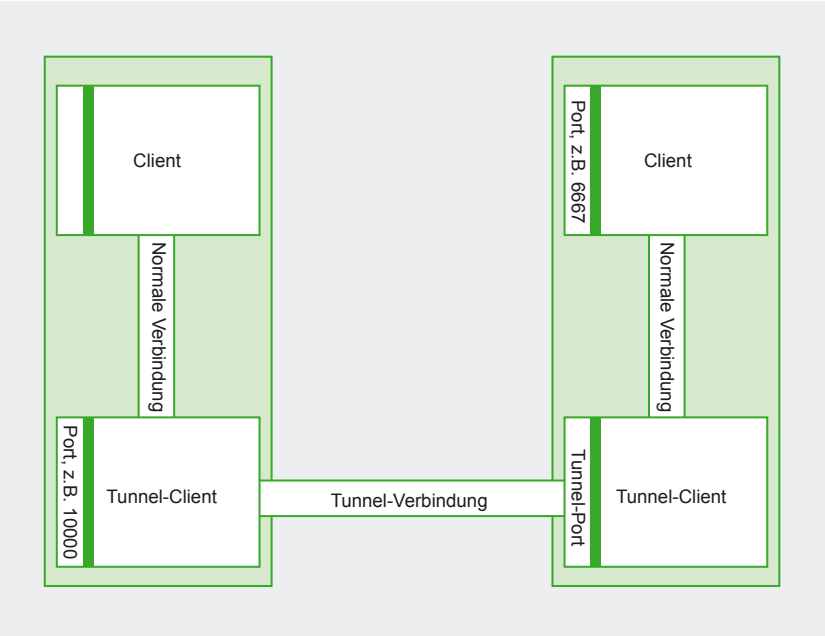

Abbildung 4. *Schema des Application-Layer tunnelings via TCP-Streamsocket*

Ports 53 für DNS, 80 für HTTP, und bei einem externen Mailsystem noch um Port 110 für POP3 (oder Port 143 für IMAP) und Port 25 für SMTP. Eventuell kommen noch Ports für die SSL verschlüsselten Varianten dieser Dienste (etwa Port 443 für HTTPS oder 993 für IMAP over SSL) mit dazu. Fakt ist: Eine einfache SSH- oder IRC-Verbindung ausserhalb ist ohne Weiteres nicht möglich (was wohl auch vom Arbeitgeber des Unternehmens so gewollt ist).

Mit Hilfe eines Tunnels lässt sich dies ändern. Ist beispielsweise ein SSH-Zugriff erlaubt (z.B. Aufgrund einer Fehlkonfiguration, oder damit der Administrator schnell aus jedem Subnetzwerk Rechner in einem anderen Netzwerk über das Internet administrieren kann), so könnte man den bereits oben beschriebenen SSH-Tunnel verwenden um eine IRC-Verbindung herzustellen. Der grosse Vorteil von SSH-Tunneln besteht in gebotenen Verschlüsselung.

Doch falls eine SSH-Verbindung nicht erlaubt ist, benötigt man andere Tunneling-Tools mit denen sich andere Protokolle tunneln lassen. Besonders häufig anzutreffen sind Tools für das HTTP-Tunneling, wie etwa *httptunnel* oder *HTTPort*.

Abbildung 5. *Eine Browser-Verbindung über den HTTP-Tunnel*

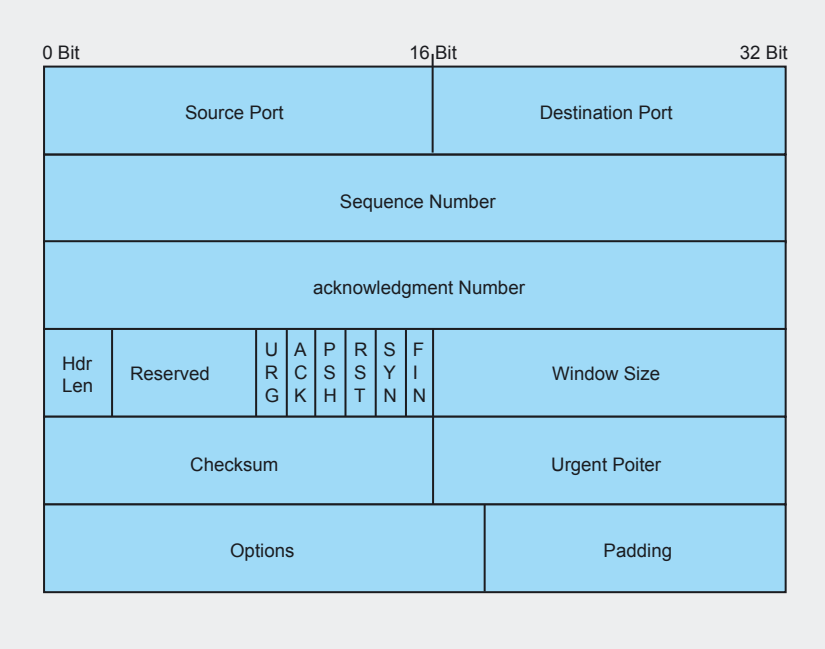

Abbildung 6. *Aufbau des TCP-Headers. Besonders die reservierten Bits sowie die beiden Bits URG und PSH sind zum Tunneling eines Bytes zu gebrauchen*

Im Folgenden – und natürlich auch zum mitmachen – bauen wir einen HTTP-Tunnel mit dem Tool *httptunnel* auf, was ebenfalls recht einfach von Statten geht.

Zunächst startet man auf dem Zielrechner, zu dem die Verbindung aufgebaut werden soll den Server `hts`. Das Zielsystem ist Port 80 auf *server.xyz*. Der Tunnel-Server nimmt auf Port 10000 die Verbindung entgegen.

```
$ hts -F server.xyz:80 10000
```

Der Client wird nun lokal gestartet und verbindet sich mit dem Server auf *server.xyz*, Port 10000. Er selbst soll auf Port 20000 die Verbindung des Browsers entgegen nehmen.

```
$ htc -F 20000 localhost:10000
```

Mit dem Browser verbindet man sich nun auf 127.0.0.1, Port 20000, wie Abbildung 5 zeigt, um den Tunnel zu verwenden. HTTP über HTTP zu tunneln ist zwar nicht sonderlich sinnvoll, aber wenn Sie auf dem Server statt Port 80, Port 22 angeben, dann könnten Sie natürlich genauso gut SSH tunneln.

```
$ ssh -p 20000 user@localhost
```

Doch was tut man, wenn man kein Tunnelingtool hat (und auch keins schreiben kann), das einen der Ports, die einem zur Verfügung stehen, nutzen kann? Das ist noch immer kein Beinbruch, denn auch hier kann man dank diverser Tools noch immer recht einfach Tunnel zwischen zwei TCP/IP-Systemen aufbauen und sogar auf Routing zurück greifen. Besonders beliebt sind hierbei ICMP-Tunnel.

vstt

Als Software für einen ICMP-Tunnel steht unter Anderem das Tool *stunnel* sowie mein Programm *vstt* zur Verfügung. Bei *vstt* handelt es sich um ein Programm, dass je nach Situation den Ausweg über einen TCP- oder einen ICMP-Tunnel ermöglicht (UDP-Tunnel sind leider noch nicht implementiert). (Ich muss wohl nicht sagen, dass dieses Programm nur für legale Zwecke verwendet werden darf...) Auf diese Weise sollten die meisten Firewalls umgangen werden können. Schließlich kann man, falls alle TCP-Ports gesperrt sind, immer noch auf ICMP zurück greifen. Ausserdem werden nicht nur blanke TCP-Tunnel, sondern auch solche mit einer Funktion zum Verstecken der Verbindung ermöglicht, dazu später mehr.

Um einen *vstt-ICMP*-Tunnel zwischen zwei Rechnern aufzubauen startet man zunächst auf beiden *vstt* im *ICMP*-Tunneling Modus. Im Folgenden soll eine SMTP-Verbindung getunnelt werden. Der Aufruf von *vstt* ist praktisch immer gleich, nur dass man statt `-p icmp` bspw. auch `-p pop3` oder `-p none` einsetzen kann. Mit `-r <Port>` wird jeweils der Port angegeben, auf dem man die Verbindung entgegen nehmen will. Mit `-t <Port>` wird der Port des Peers angegeben. Für ICMP benötigt man eigentlich keine Ports, doch Sie dienen in diesem Fall zur Identifikation der Verbindung für die Sockets. Die eigene Adresse wird letztlich noch mit `-a`, die des Peers mit `-m` angegeben.

```
client# vstt -p icmp -r 10002 \
-t 10001 -a 192.168.2.20 \
-m 192.168.2.22

server# vstt -p icmp -r 10001 \
-t 10002 -a 192.168.2.22 \
-m 192.168.2.20
```

Nun sollte der ICMP-Tunnel bereits stehen. Als nächstes kommt das Tool `s2f` zum Einsatz, das Bestandteil von *vstt* ist. Es verbindet den lokal laufenden *vstt*-Prozess mit einem Socket und ermöglicht so die Verbindung zwischen Client und Server.

Auf dem SMTP-Server starten wir `s2f` im Client-Mode, was bedeutet, dass `s2f` sich als Client mit dem lokalen SMTP-Port (25) verbindet. Auf dem Client hingegen starten wir `s2f` als Server (Parameter `-s`), damit sich der lokale Mail-Client mit `s2f` verbinden kann.

```
server# s2f -p 25
client# s2f -s -p 20000
```

Ab jetzt muss nur noch mit einem Mail-Client auf dem Client an Port 20000 verbunden werden, und die SMTP-Session kann beginnen.

```
client$ telnet localhost 20000
Trying 127.0.0.1...
Connected to amilo.sun.
Escape character is '^]'.
220 eygo.sun ESMTP Sendmail
8.13.8/8.13.8; Mon, 25 Sep 2006
22:37:48 +0200 (CEST)
...
```

Tunneln in jeder Situation

Jetzt wird es wieder Zeit für Theorie. Fakt ist, dass man praktisch durch jedes Protokoll Daten tunneln kann. Bei einigen Protokollen ist die Realisierung eines Tunnels sehr aufwendig, bei anderen sehr einfach. Im Folgenden bespreche ich gängige sowie einige sehr unübliche Tunneling-Techniken, die aber zweifellos möglich sind und mit denen teilweise die übliche TCP/IP-Landschaft verlassen wird und man durch praktisch jede Firewall kommt, wenn man vor dieser und hinter dieser ein System benutzen kann. Einige dieser Angriffe funktionieren nur im lokalen Netzwerk, weil Low-Level Protokolle benutzt werden, die nicht routingfähig sind.

POP3

Um einen Eindruck davon zu bekommen, wie leicht es ist, Daten zu tunneln, soll das POP3-Protokoll als erstes Beispiel dienen. In diesem Protokoll gibt es diverse Befehle wie etwa `RETR n`.

Dieser Befehl holt die Nachricht mit der Nummer `n` ab. Das ist für jeden Administrator, der seine Verbindungen überwacht ein ganz normaler Anblick. Doch selbst hier kann man Daten bereits versteckt tunneln. Dies geht indem man beispielsweise - wie ich es in *vstt* implementiert habe - eine pseudo-POP3-Verbindung aufbaut, die eigentlich keine ist. Für jedes zu übertragene Byte wird der Befehl `RETR n` ausgeführt, wobei aber `n` keine Nachrichten-Nummer, sondern der ASCII-Wert des zu übertragenen Bytes ist. Der *vstt*-Aufruf ist Analog zur ICMP-Variante, nur dass nun POP3 als zu benutzendes Protokoll auf dem Client und dem Server angegeben werden muss. Hier daher nur das Beispiel für den Client:

```
client# vstt -p pop3 -r 10002 \
-t 10001 -a 192.168.2.20 \
-m 192.168.2.22
```

Eine weitere Möglichkeit wäre es, jeweils 4 Bit eines Bytes zu übertragen indem man die Zeichen `RETR` als Binärwerte verwendet. Ist ein Zeichen groß geschrieben, steht es für eine *1*, ein kleines Zeichen hingegen repräsentiert eine *0*. Würden also die zwei Befehle `reTR 4\r\nReTR 3\r\n` aufeinander folgen, dann würde dies das Byte 00111011 repräsentieren. Möchte man es noch weiter treiben, so kann man jeweils nur 1 Bit übertragen, indem man entweder nur 1 oder 2 Leerzeichen zwischen dem Befehl und der Nachrichtennummer überträgt, wobei eben wieder eines für eine *1* und zwei für eine *0* stehen (oder umgekehrt). Solche Tunnel sind äußerst schwer zu detektieren. Falls man allerdings jeweils nur ein oder vier Bit überträgt, so müssen extrem viele POP3-Pakete gesendet werden, was dann wiederum sehr auffällig sein kann, weil die Traffic-Rate – je nach Umgebung – deutlich ansteigen kann. Auf die selbe Weise lassen sich natürlich auch die meisten anderen Application-Layer Protokolle wie SMTP, NNTP, IMAP, HTTP, FTP etc. verwenden, um Daten zu verstecken und gleichzeitig zu tunneln.

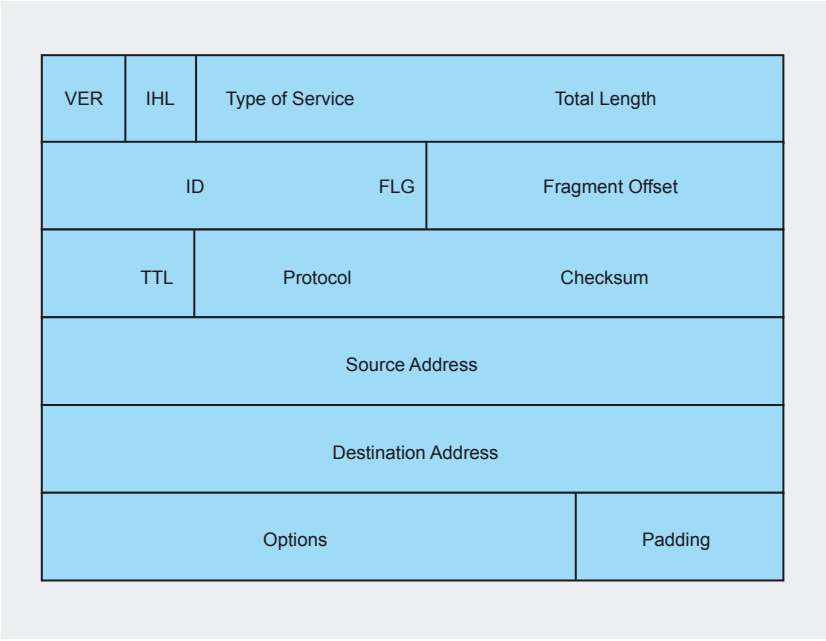

Abbildung 7. *Aufbau des Ipv4-Headers*

TCP

TCP basierte Streamsocket-Tunnel kennen Sie bereits durch das erste Beispiel (SSH) und den soeben beschriebenen POP3-Tunnel. Doch sind diese nicht die einzigste Möglichkeit durch TCP zu tunneln. Im TCP-Header können einige Bits dazu benutzt werden, um Daten darin zu verstecken.

Simmuliert man beispielsweise eine Verbindung zwischen zwei Systemen, dann werden einige Bits im TCP-Header gar nicht, und andere nur bedingt verwendet. Besonders interessant sind die 6 reservierten Bits, die an sich keine weitere Verwendung finden. Das darauf folgende Urgent-Bit könnte man noch als siebentes Bit verwenden. Doch Achtung: Ist das Urgent-Bit gesetzt, wird auch der Urgent-Pointer ausgewertet, der das Ende der als dringlich markierten Daten im Payload zeigt. Um ein einzelnes Byte zu übertragen fehlt nun nur noch das 8. Bit. Dass auf das Urgent-Bit folgende ACK-Bit sollte man möglichst

unangetastet lassen, da dieses eine zu wichtige Bedeutung im TCP-Traffic hat, Es wird verwendet, um empfangene Daten zu bestätigen. Daher sollte man erst das 3. Bit der TCP-Flags beanspruchen: das PUSH-Flag. Dieses Bit wird verwendet um einzelne TCP-Segmente unmittelbar an den Applicationlayer weiterzuleiten, ohne dass erst eine bestimmte Anzahl von Bytes (dafür gibt es übrigens konfigurierbare Watermarks) empfangen werden muss. Einige Anwendungen wie SSH und Telnet benutzen übrigens generell das PUSH-Flag, um jeden Tastendruck sofort zu übertragen und auszuwerten.

IPv4 und IPv6

Um Daten direkt im Ipv4 oder Ipv6 Protokoll zu tunneln gibt es auch einige Möglichkeiten. In Ipv4 könnten Bits zur Flussoptimierung, oder auch einige Bits des Wertes Time-to-live (TTL) missbraucht werden. Der TTL-Wert ist 8 Bit groß. Wird ein IP-Datenpaket versendet, so ist der TTL-Wert zu begin 0xff. Jeder Router, den das Paket passiert, dekrementiert den TTL-Wert um exakt 1. Erreicht der TTL-Wert die Zahl 0, so wird das Paket verworfen. Man verwendet den TTL-Wert unter Anderem, um unendlich lange Routing-Loops zu verhindern, in denen Pakete unendlich kreisen würden. Weiß man aber, dass zwischen zwei Tunnelendpunkten bspw. nur 2 Router (man spricht dann von 2 Hops) liegen, und dass keine andere Route zwischen den beiden Hops besteht, so kann man davon ausgehen, dass die ersten 6 Bits nicht manipuliert werden, und diese nach belieben zum tunneln verwenden. Diese Technik ist allerdings sehr umständlich, individuell anzupassen und sehr fehleranfällig. Mit etwas Kreativität könnte man sich ansonsten auch die optionalen Ipv4-Optionen zu Nutzen machen.

In Ipv6 könnte man hingegen die 4 Bits des Priority-Feldes verwenden um ein halbes Byte zu tunneln. Ebenfalls denkbar ist die modifikation des Hop-Limits, dass den Nachfolgeder des Ipv4 TTL-Wertes darstellt.

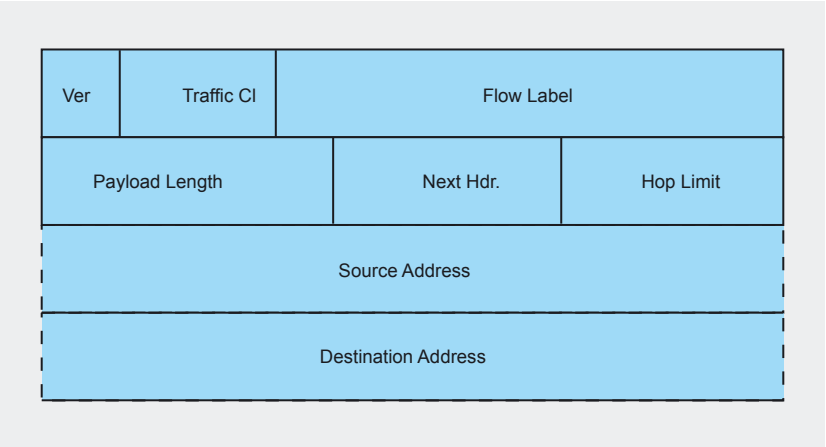

Abbildung 8. *Aufbau des Ipv6-Headers*

ICMP

Beim ICMP-Protokoll gibt es verschiedene Möglichkeiten, Daten zu tunneln. Man muss hierzu wissen, dass der ICMP-Header sich je nach ICMP-Typ und ICMP-Code unterschiedlich gestaltet und eine unterschiedliche Bedeutung aufweist.

Am Einfachsten ist es ICMP-Daten über ICMP-Echo Datagramme zu tunneln. In echo-Requests ist Platz für sehr viele Daten, weshalb ein Ping-Tunnel auch die beste Möglichkeit ist, Daten über ICMP zu tunneln. Hier gibt es allerdings etwas zu beachten: Würden Sie in einem Netzwerk wie meinem versuchen ICMP-Tunnel aufzubauen, würden Sie scheitern. Warum? Weil viele Administratoren und andere komische Leute üblicher Weise ICMP Ping-Requests (also Ping-Anfragen) blocken. Was hingegen nicht blockiert wird sind Ping-Response Nachrichten. So ist es möglich, einen Ping-Request in andere Netzwerke zu schicken, und die Antwort zu erhalten, ohne dass Hosts aus anderen Netzwerken Ping-Requests in das eigene Netzwerk senden können. *vstt* benutzt desshalb ICMP Echo-Response Pakete (ICMP-Type 0) statt Echo-Request Pakete (ICMP-Type 8), um ICMP-Tunnel aufzubauen. In der Default-Konfiguration können dabei bis zu 64 Byte pro Paket versendet werden. Wenn man möchte, kann man aber auch weitaus mehr Daten pro Paket verpacken.

DNS

Eine recht schwierige Möglichkeit Daten zu tunneln (speziell wenn man ganze IP-Pakete tunneln möchte) ist, diese durch das Domain Name System (DNS) zu schleusen. Dazu benötigt man einen Fake-DNS Server. Die Anfrage wird in der Form *ABC.mydomain.com* vom Client an den Server (oder einen anderen Server im Internet, der dann den eigenen Server anfragen muss) gestellt. Der Teil ABC des Domainnamens steht dabei für die DNS-Encodierte Form der zu übertragenen Daten. (Mehr zur Encodierung der DNS-Requests finden Sie im DNS-RFC oder Lesen Sie den Source Code meines DNS-Servers Xyria:DNSd).

Ein DNS-Header kann aus bis zu 127-Level von je 127 Zeichen langen (Sub-)Domains bestehen und kann somit recht gut Daten übertragen. Bedenken Sie jedoch, dass normale DNS-Requests über UDP-Pakete laufen und diese in Verwendung mit dem DNS nur 512 Byte groß sein dürfen. Ziehen Sie von diesen Bytes den DNS-Header und den Platz, den die Encodierung beansprucht ab, dann erhalten Sie, je nachdem wie gut ihre Encodierung geschrieben ist, deutlich weniger, aber immernoch genug Platz. Wichtig ist, dass immer TXT-Records angefragt werden. Dadurch kann der Server mit einem TXT-Record antworten, in dem die Encodierung viel einfacher

ist, und daher können auch einfacher und schneller mehr Daten übertragen werden. Allerdings muss man sicherstellen, dass wirklich immer der eigene Pseudo-DNS Server von den dazwischen geschalteten Servern abgefragt wird, was sich ebenfalls besonders schwierig realisieren lässt – zum Beispiel weil gleiche Anfragen im Cache von dazwischengeschalteten DNS-Servern landen könnten (und somit die Verbindung kompromittieren könnten). Zu diesem Zweck müsste – unter Verwendung einer Chiffrierung zur generierung von immer wieder unterschiedlichen DNS-Namen – sich jede Seite alle Requests merken und bei doppelten Requests mit einem neuen Schlüssel, der in einem Sub-Protokoll enthalten seien könnte, einen neuen Wert generieren, den der Peer-Endpunkt mit dem natürlich zu übertragenden Schlüssel dechiffrieren kann. Wenn Sie nun trotz Allem denken, es sei relativ einfach, Daten über DNS zu tunneln, dann lesen Sie den letzten Abschnitt des Artikels zum Thema Reliability.

CDP

Für den absurden Fall (das Absurde ist schließlich interessant!), dass man gar keine TCP/IP-Daten versenden kann, ist es in Sonderfällen (z.B. weil TCP/IP nicht in den Kernel einkompiliert ist und ein Netzwerk nur mit AppleTalk oder IPX läuft) immer noch möglich, Daten durch andere Protokolle zu tunneln. Diese sind in der Regel jedoch nicht routingfähig, weshalb man Daten praktisch nur direkt austauschen kann (und sich einen Proxy selber schreiben müsste... aber jetzt geraten wir für normaldenkende Sicherheitsverantwortliche in den Bereich des Absurden). Mögliche Kandidaten hierfür sind das Cisco Discovery Protocol (CDP), die IPX- und AppleTalk-Protokolle sowie einige selbst gebastellte Protokolle. Eine weitere Möglichkeit bestünde darin, in einem Netzwerk, dass das Spanning Tree Protocol (STP) nicht benutzt, mittels dafür geschriebener Programme genau in diesem Protokoll Daten zu verschicken. Anstatt einer Adresse könnte man in dann die zu tunnelnden Daten verstecken. Doch nun genug mit diesem Ausblick und zurück zum letzten wichtigen Thema.

Weitere Möglichkeiten

Wenn Sie nun denken, die Möglichkeiten des Tunnelings seien durch diese paar Protokolle und die gesamten Application-Layer Protokolle erschöpft, dann irren Sie sich. Beispielsweise könnte man recht gut in dynamischen Routingprotokollen tunneln. Im Routing Information Protocol (RIP) sind beispielsweise ganze 16 Bit ungenutzt. Würde man folglich die für ein Netzwerk typischen Routing-Meldungen zwischen einigen Routern generieren, so dass dem Administrator nichts daran auffällt, und würde man zudem Sniffende Tunnelendpunkte haben, die den Datenverkehr zwischen solchen Routern mitlesen, dann könnte man sehr wohl unbemerkt kommunizieren. Durch Routing-Protokolle sollte man allerdings nur im Extremfall tunneln, da ein kleiner Fehler grobe negative Auswirkungen auf ein Netzwerk haben kann!

Reliability

Leider gibt es da nämlich noch ein Problem (sofern man nicht grad eine TCP-Streamsocket Verbindung verwendet): Pakete können unterwegs verloren gehen, in verschiedener Reihenfolge beim Tunnelendpunkt ankommen oder Schaden nehmen. Im lokalen Subnetz kann man durchaus tunneln, ohne sich um soetwas kümmern zu müssen, doch wenn man Daten über das Internet versenden möchte, dann kommt man nicht um einen Algorithmus herum, der einen Schutz implementiert. Die Vorgehensweise hierfür ist natürlich sehr protokollspezifisch. In *vstt* habe ich für die ICMP-Tunnel jedem Paket eine ID verpasst. Diese ID muss von der Gegenstelle ebenfalls mit einem ICMP-Paket bestätigt werden. Wird die ID nicht bestätigt, dann wird das Paket erneut gesendet.

Ausserdem kümmert sich *vstt* darum, dass die Pakete, die zu groß für ein einzelnes ICMP-Paket sind, auf mehrere aufgeteilt werden und am Ende wieder zusammengesetzt werden. Erst wenn alle Teile eines ursprünglichen Pakets übertragen wurden, wird der Payload an s2f weiterleitet. So oder ähnlich funktionieren die meisten Tunneling-Tools die direkt mit ICMP oder TCP tunneln. ●

Im Internet

- *http://www.tcp-ip-info.de/tcp_ip_und_internet/tunneling_protokolle.htm* - weitere VPN-Protokolle;
- *http://en.wikipedia.org/wiki/Tunneling_protocol* sowie *http://de.wikipedia.org/wiki/TCP/IP-Referenzmodell* – Wikipedia-Artikel;
- *http://www.nocrew.org/software/httptunnel.html* – httptunnel;
- *http://www.stunnel.org/* - stunnel;
- *http://doomed-reality.org* - vstt;
- *ftp://ftp.rfc-editor.org/in-notes/rfc1035.txt* - RFC 1024: P. Mockapetris: Domain Names – Implementation and Specification, Nov. 1987.

Literatur

- *Praxisbuch Netzwerksicherheit*, Galileo-Press, 2005 - S. Wendzel & J. Plötner.

Über den Autor

Steffen Wendzel ist 21 und beschäftigt sich seit vielen Jahren mit der Sicherheit von Unix-Systemen und TCP/IP-Netzwerken. Er entwickelte diverse OpenSource Software, ist Autor mehrerer Bücher zu den Themen Linux und Netzwerksicherheit, Security Consultant bei Plötner-IT sowie Student der Informatik an der FH-Kempten (Deutschland). Seine Webseite: *http://cdp.doomed-reality.org*