TCP / IP

Beitrag von Vimes
Alle 6 Lektionen

Vorwort:
Da das Thema ziemlich groß ist, habe ich es in 6 Lektionen zerlegt.

Teil 1: Einleitung

Warum überhaupt diese Lektion? Zum einen ist sie Selbstzweck - das Thema ist nämlich im Grunde ziemlich spannend (auch wenn es mir am Anfang nicht so vorkam). Zum anderen bildet sie die Grundlage für die Themen "Personal Firewalls" und "Sicherheit im Heimnetzwerk". Meldungen wie "Das Programm XYZ versucht, über Port 2345 auf die IP 127.0.0.1 zuzugreifen" , sind sonst nicht ohne weiteres verständlich.

Kommunikation, aber wie?

Wie reden Rechner eigentlich miteinander?
An der Spitze stehen die Anwendungsprogramme wie z.B. ein Browser, das Mailprogramm oder ein FTP-Client.

Ob ich nun eine Webseite aufrufe, eine E-Mail versende oder über FTP eine Datei hochlade, in allen diesen Fällen versucht das Programm nun, mit einem anderen Rechner zu reden, um ihm zu übermitteln, was ich von ihm möchte.

Stufe1: Programm
Die unterschiedlichen Programme sprechen unterschiedliche Sprachen oder Protokolle. Für den Aufruf von Webseiten wird HTTP benutzt, für verschlüsselte Verbindungen HTTPS, zum Abrufen von E-Mails wird POP3 gebraucht und für das Versenden von E-Mails SMTP.

Nehmen wir den Browser. Ich möchte, daß er die Seite "www.computerproblemboard.de" öffnet. Der Browser formuliert meine Anfrage so, daß der Server auf der anderen Seite sie verstehen kann, und sagt dem Betriebssystem, daß er gerne Daten versenden möchte. Das muß zentral gesteuert werden, damit mehrere Programme gleichzeitig Daten senden und empfangen können. Das geht nur dann gleichzeitig, wenn eine zentrale Stelle die Zugriffe nach außen koordiniert.

Stufe 2: TCP und UDP
Das Betriebssystem nimmt die Anfrage des Browsers entgegen. Dabei sagt der Browser, womit er reden möchte und auf welcher Unteradresse. Das läuft wie in einem Wohnheim: Ich schreibe auf einen Brief nicht nur die Hausnummer, sondern auch die Wohnungsnummer.
Die Daten werden nun in Häppchen aufgeteilt und so verpackt, daß immer drinsteht, wer mit wem reden möchte und unter welcher Wohnungsnummer.

Stufe 3: IP
Die TCP- oder UDP-Pakete werden nun an das Internet Protokoll weitergegeben. Auf dieser Ebene werden die Daten zum Webserver übertragen.

Stufe 4: Hier folgt streng genommen noch das letzte Einpacken, nämlich in die Protokolle, die vom Internet-Anbieter "gesprochen" werden, z.B. PPP oder PPPoE. Das ganze geht über ein Kabel oder Funk, ist ja klar. Dieser letzte Schritt soll uns hier nicht weiter interessieren.

Den ganzen Vorgang kann man sich wie bei diesen russischen Puppen vorstellen, bei denen in jeder Puppe immer eine noch kleinere drinsteckt:

| Netzwerk | IP | TCP | HTTP | (für den Fall, daß ich eben den Browser benutze).

Auf der Empfängerseite wird das ganze dann ausgepackt: Zuerst auf Netzwerkebene, dann auf IP-Ebene, dann auf TCP-Ebene und zuletzt wird vom Betriebssystem dem Webserver das HTTP-Päckchen übergeben. Der wertet meine Anfrage aus, sucht die Seite raus und sendet sie in Häppchen zurück.

(Das ganze habe ich bewußt ziemlich vereinfacht, man möge es mir bitte nachsehen)

Teil 2: Client / Server

Bevor wir tiefer in das eigentlich Thema TCP/IP einsteigen, müssen wir uns zunächst die im ersten Teil gezeigte Kommunikation noch einmal genauer anschauen.

Wenn eine Software auf meinem Rechner mit einer Software auf einem anderen Rechner kommunzieren möchte, dann liegt einer von zwei Fällen vor:

1. Mein Rechner fragt bei dem anderen Rechner Daten ab.
2. Der andere Rechner fragt von meinem Rechner Daten ab.

Im 1. Fall handelt es sich bei mir um einen Client.
Im 2. Fall ist es bei mir ein Server.

Der Unterschied ist folgender:

Ein Client sendet über das Betriebssystem (dieser Teil wird auch "TCP/IP-Stack" genannt) Anfragen an einen Server, z.B. mein Firefox an den Server, auf dem www.computerproblemboard.de liegt.

Der TCP/IP-Stack weiß, daß es sich beim Firefox um einen Client handelt. Deswegen reicht er an den Firefox nur solche Pakete (= Daten) weiter, die auf seine Anfrage passen.
Ein Client nimmt also nur Antworten auf Fragen entgegen. Hat er keine Frage gestellt oder passt eine Antwort nicht zu der von ihm gestellten Frage, dann wirft der TCP/IP-Stack diese Antwort schlicht und einfach weg. Der Sender wird darüber informiert.
Beispiele für Clients: Browser, Mailprogramme, FTP-Programme zum Herunterladen von einem FTP-Server, Instant-Messenger-Programme (mit Einschränkungen).

Ein Server dagegen nimmt Anfragen entgegen und antwortet darauf. Von wem diese Anfrage kommt, ist zunächst einmal völlig egal. Alle Pakete, die die Adresse des Servers tragen (dazu später mehr) werden vom TCP/IP-Stack an den Server durchgereicht.
Ein Server ist also, solange keine Sicherheitsmaßnahmen getroffen werden, von Krethi und Plethi zu erreichen (oder von Hans und Franz).
Beispiele für Server: Webserver wie Apache und IIS (Microsoft), FTP-Server, SAMBA-Dateiserver.

Wichtig:
Eine Sonderrolle nehmen die Dienste unter Windows ein, von denen viele bestimmt schon gehört haben. Dazu gehört z.B. der berüchtigte RPC-Dienst, über den der Blaster hereinkam.
Diese Dienste sind technisch gesehen ebenfalls Server, d.h. sie nehmen jede Anfrage entgegen. Sie arbeiten mit Systemrechten, d.h. sie dürfen auf dem betroffenen System alles.

Wir halten fest:
Clients können aus dem Internet heraus schlecht angegriffen werden, da sie nur Antworten auf ihre eigenen Fragen zugestellt bekommen.

Server dagegen nehmen prinzipiell jede Anfrage aus dem Internet entgegen. Weist der Server nun einen Fehler auf, kann man über geeignete Anfragen (Pakete) den Server dazu bringen, daß er Dinge tut, die man eigentlich nicht möchte, z.B. Programme ausführen, die ein Angreifer dem Server zuschiebt.

Achtung:
Wir müssen nun noch zwei Fälle unterscheiden.
Im ersten Falle habe ich bewußt einen Server am Laufen, z.B. einen Webserver, um meine private Homepage anzubieten. Das ist ok, solange ich mir über die Sicherheitsrisiken im klaren bin.
Im zweiten Falle läuft auch auf meinem Rechner ein Server, von dem ich aber nichts weiß. Dieser Server nimmt Anfragen seines eigentlichen Herren und Meisters entgegen. Anders ausgedrückt: Der Server nimmt Befehle entgegen.
Mögliche Folgen: Diebstahl persönlicher Daten, Weiterverbreiten des Servers an andere, ahnunglose User (bzw. deren Rechner), Angriffe auf andere Rechnersysteme.
Bei einem solchen Server handelt es sich also um Schadsoftware.

Das Problem ist nun, daß solche Server selten offen arbeiten, so daß man sie einfach erkennen und beseitigen könnte. Häufig tarnen sich diese Programme, so daß die Suche danach aufwendig und für den Laien idR nicht durchführbar ist. Da zudem die Gefahr besteht, daß nach Belieben Programme "nachgeladen" und installiert werden, ist ein System, auf dem ein fremder Server läuft, nicht mehr vertrauenswürdig. Es ist sofort vom Netz zu trennen und neu aufzusetzen.

Teil 3: IP-Adressen und Ports

Im ersten Teil haben wir gesehen, wie die Kommunikation zwischen zwei Rechnern funktioniert.

Damit nun mein Browser eine Anfrage an das Board stellen kann, muß mein Betriebssystem, das die Anfrage ja verarbeitet, zwei Dinge wissen:

1. Welcher Rechner ist überhaupt gemeint?
2. Welches Programm auf diesem Rechner soll die Anfrage entgegennehmen?

Das entspricht in etwa der Straße mit Hausnummer und dem Namen des Empfängers.

Punkt 1 meint die IP-Adresse eines Rechners. IP steht für Internet Protokoll. Jeder Rechner, der am Internet [1] hängt, bekommt eine solche Adresse zugewiesen, die eineindeutig (=einmalig, kein Tippfehler), ist.
Für das Board ist das z.B. die 62.75.252.213 . Damit weiß das Betriebssystem, an welchen Rechner es die Anfrage schicken soll, aber noch nicht, welches Programm dort das Paket erhalten soll.
Hier kommt Punkt 2 ins Spiel, wir sagen dem anderen Rechner auch noch, welchen der in ihm ansässigen Herrschaften (= Programme, genauer: Server) wir zu sprechen wünschen. Dazu geben wir den sogenannten Port an.
In unserem Beispiel läuft ein Webserver wie das Computerproblemboard üblicherweise auf Port 80.
Jetzt weiß der TCP/IP-Stack, das ist der Teil des Betriebssystems, der solche Vorgänge verwaltet, so er die Anfrage hinsenden muß.

Das ganze funktioniert umgekehrt auch für einen Server. Der Rechner, auf dem der Server läuft, hat eine IP-Adresse, über die der Rechner aus dem Internet erreichbar ist. Das Porgramm an sich - der Server - braucht aber eine eigene Unteradresse, damit der TCP/IP-Stack an ihn alle Anfragen durchreichen kann. Also bekommt der Server vom Betriebssystem einen Port zugewiesen. An diesem "lauscht" der Server dann.

Zusammengefaßt:
Ein Port ist eine Adresse für ein Programm, sei es ein Client oder ein Server, auf einem Rechner.
Ein Client sendet Anfragen und bekommt nur dazugehörige Antworten zu sehen. Ein solcher Port wird als geschlossen bezeichnet, da er nicht "lauscht".
Ein Server dagegen nimmt Anfragen aus dem Internet entgegen. Ein solcher Port wird als geöffnet bezeichnet, da der Server ja auf Anfragen wartet.

Jetzt liegt die Frage nahe: wieviele Ports gibt es überhaupt und sind die irgendwie bestimmten Programmen oder Programmtypen fest zugeordnet?

Es gibt 65535 Ports, von 1-65535. Davon sind die Ports 1-1023 sog. well known ports (bekannte Ports). Sie dürfen nur mit Systemrechten geöffnet werden, ein eingeschränkter Benutzer (bzw. seine Programme) darf das nicht.
Die Ports darüber können auch von Nicht-Administratoren geöffnet und benutzt werden.

Einige Ports und ihre zugehörigen Protokolle:
21 FTP (Dateiübertragung)
23 Telnet (Fernzugriff)
25 SMTP (Mailversand)
80 HTTP (Webserver)
110 POP3 (Mailabruf)

Wichtig: Es gibt zwar Tabellen wie obige, die Ports Programmen zuordnen. Aber das ist kein in Stein gemeißeltes Gesetz. Ein bösartiger Server (Schadprogramm) kann durchaus auch auf Port 80 oder 110 funken - ohne deshalb ein Webserver oder ein Mailclient zu sein.

Teil 4: TCP/IP

TCP ist ein verbindungsorientiertes Protokoll. Das heißt, der Client baut zum Server eine Verbindung auf, die Daten werden nach dem Senden quittiert, so daß man Übertragungsfehler erkennen und korrigieren kann und am Schluß wird die Verbindung beendet, so daß der Server weiß, daß nun keine Anfragen mehr kommen.

Das funktioniert so:

Der Client (z.B. ein Browser) reicht seine Anfrage beim TCP/IP-Stack ein. Der Stack weist ihm einen Port zu, über den dann die Anfrage rausgeschickt wird.

Das kann man sich vorstellen wie bei einem Brief: Auf den Umschlag wird der Absender mit allen notwendigen Angaben geschrieben. Also meine IP-Adresse und der Port, über den die Anfrage hinausgeht.
Ebenso kommt auf den Umschlag der Empfänger, d.h. die IP-Adresse des Servers und sein Port.

Die Verbindung wird nun aufgebaut, indem Client und Server ein Frage-Antwort-Spiel ablaufen lassen:

Client: "Bist Du da? Ich möcht was wissen"
Server: "Kann Dich hören. Bin da."
Client: "Super. Hab Dich verstanden. Sende jetzt Daten."

Die Verbindung steht damit. Der TCP/IP-Stack sendet nun meine Anfrage raus.
Dabei werden die Daten so verpackt, daß der Server auf der Empfängerseite immer sehen kann, wie er sie wieder zusammenzusetzen hat, damit das Ganze einen Sinn ergibt.

An einem Beispiel:
Mein Browser möchte die Startseite des Problemboards haben.
Als Anfrage soll dann rausgehen:
"Zeig mir die Startseite des Problemboards".

Diese Anfrage ist so zu groß, sie muß zerlegt werden[1]. Wir erhalten dann:

[Ziel Port 80][1][Zeig]
[Ziel Port 80][2][mir]
[Ziel Port 80][3][die]
[Ziel Port 80][4][Startseite]
[Ziel Port 80][5][des]
[Ziel Port 80][6][Problemboards]

[1] Es komme mir jetzt bitte keiner, daß das typische TCP-Paket solche Inhalte locker transportieren kann, ich bin zu faul, mir ein geeignetes Beispiel zu überlegen, das auch die Profis zufriedenstellt...

Egal, in welcher Reihenfolge diese Päckchen ankommen, der Server kann sie wieder in der richtigen Reihenfolge zusammensetzen, da sie ja numeriert sind. (Natürlich ist's in der Realität ein wenig komplexer, aber wir wollen ja nur einen Überblick.)

Der Server (hier: das Problemboard) sendet dann die gewünschten Daten zurück. Das funktioniert im Prinzip genauso wie oben. Sender und Empfänger quittieren jeweils den Empfang, so, wie man beim Postboten ein Einschreiben quittieren muß.

Ganz am Schluß sagt der Client dem Server noch einmal explizit, daß er fertig ist. Die Verbindung wird dann wieder abgebaut. Das ist so, wie wenn man nach einem Telefonat auflegt.

Teil 5: UDP / IP

Gerade haben wir gesehen, daß TCP verbindungsorientiert ist.
UDP dagegen funktioniert anders. Hier "brüllt" der Client seine Anfragen einfach hinaus, ohne

a) vorher ausdrücklich eine Verbindung aufzubauen
b) Daten zu quittieren oder
c) seinerseits eine Quittung zu erhalten.

Die Anfragen gehen wie bei TCP natürlich an eine IP und einen Port, aber ansonsten läuft das in etwa so:

Client: "Hallo, ich will was von Dir." (Wartet keine Antwort ab)
Client: "Ich will das..."
Server: "Bitte sehr..."
Client: "Und das..."
Server: "Bitte sehr..."
Client: "Und das..."
Server: "Bitte sehr..."

Am Schluß legt die Rotzgöre (der Client) einfach auf, ohne sich groß zurückzumelden.

UDP ist damit ein vergleichweise einfaches Protokoll.

Wichtig:

Bei UDP ist die Endanwendung selbst (d.h., z.B. der angesprochene Server) dafür zuständig, sich aus dem ganzen Mist einen Sinn zusammenzureimen. Denn die Numerierung der Häppchen, wie sie bei TCP erfolgt, entfällt. Außerdem bekommt man keine Rückmeldung, ob etwas verlorenging - und kann das auch schlecht erraten, daß man ja keinerlei Hinweise darauf bekommt (keine Numerierung, keine Quittungen...).

UDP wird für Aufgaben benutzt, bei denen es nicht so sehr darauf ankommt, ob alle Daten überhaupt ankommen und es dem Absender egal ist, ob er dieselbe Anfrage einmal oder zwanzigmal stellen muß (weil etwas unterwegs verlorengegangen ist).

In der Regel werden über UDP daher sehr kleine Inhalte übertragen, so daß jedes Häppchen eine in sich abgeschlossene Einheit darstellt.

UDP wird typischerweise genutzt für:

- Namensauflösung im WWW, d.h. ich tippe im Browser www.computerproblemboard ein und es wird eine Anfrage an einen Server gestartet, welche IP sich dahinter verbirgt. Das geht über UDP Port 53. Das Protokoll heißt DNS.

- Der NFS-Fileserver unter Linux läuft über UDP, um die Geschwindigkeit zu erhöhen - es ist keine ständige Verbindungskontrolle nötig.

- Viele Online-Spiele laufen nicht nur über TCP/IP, sondern auch über UDP/IP. Das ist ein guter Weg, um Steuernachrichten zwischen Client und Server zu übertragen, ohne eine weitere TCP-Verbindung aufbauen und offenhalten zu müssen.

- Multimedia-Anwendungen, die über das Internet kommunizieren, wie z.B. Netmeeting.

Teil 6: Schlußbemerkungen / Fragen und Antworten

Damit sind wir am Ende der Lektion "TCP/IP" angelangt.
Stellt sich vielleicht noch folgende Frage:

Was hat das ganze mit Sicherheit zu tun? Ist doch ein Sicherheitsworkshop hier...

1. Ohne Kenntnisse über die Art und Weise, wie Rechner sich unterhalten, habe ich keine Ahnung, welchen Gefahren mein Rechner im Internet überhaupt ausgesetzt ist.

Erst, wenn ich weiß, was einen Client von einem Server unterscheidet, weiß ich überhaupt, daß ich das eine bedenkenlos benutzen kann, das andere aber Sachverstand und Sicherheitvorkehrungen erfordert, wenn ich nicht Ziel für diverse Möchtegern-Hacker werden möchte.

2. Ohne solche Kenntnisse kann ich keine Personal Firewall und auch keinen Router sinnvoll einsetzen, geschweige denn, meinen Rechner gescheit konfigurieren (z.B. durch das Abschalten von Diensten).

Wer schon einmal hinter einem Router spielen wollte, kennt das Problem: Das Spiel geht nicht. Warum nicht? Was muß man dagegen tun? Mit Begriffen wie "Portforwarding" kann man erst dann etwas anfangen, wenn man überhaupt weiß, was ein Port ist und was er tut.

Zum Schluß noch eine Anekdote aus dem Usenet.
Fragestellung war, wie man die Sicherheit eines Apache-Webservers bezüglich Hacker-Angriffen testen könnte.

"Man baue eine 160 GB-Platte ein, installiert darauf den Apache und setzt dann in einem einschlägigen IRC-Channel [Chat-Kanal, Anm. von Vimes] die Botschaft ab: 'Achtung, Webserver mit schöner, großer Platte unter $IP-Adresse, sieht ungesichert aus, schnappt ihn Euch!!!'
Nach 24 Stunden schaue man wieder nach dem System. Ist das System sicher, ist es gut, im anderen Falle baut man die Platte aus und freut sich über 160 GB an Warez, MP3s, Nacktbildern und Filmen..."
(c) Urs Tränkner in de.comp.security.misc

MfG
Vimes