SSH

1 Grundlagen

SSH (Secure Shell) ermöglicht eine verschlüsselte Kommunikation zwischen zwei Computern. OpenSSH ist eine freie Implementation des SSH-Protokolls. In diesem Projekt verwenden wir OpenSSH for Windows.
Um eine SSH-Verbindung aufzubauen, muss auf einem Computer ein SSH-Server (Deamon) laufen und auf eingehende Verbindungen hören. In der Regel hört er auf Port 22. Um von einem Client-PC eine Verbindung zum Server herzustellen, braucht man einen SSH-Client. In diesem Projekt verwenden wir dazu PuTTY.

2 Authentifizierung

Um sich auf dem Server anzumelden, gibt es zwei verschiedene Arten für die Authentifizierung. Die erste ist mit Benutzername und Passwort, die zweite mit einem Schlüsselpaar. Dafür muss auf dem Server ein Public-Key vorhanden sein. Der SSH-Client braucht einen privaten Schlüssel.

3 Installation von OpenSSH for Windows

OpenSSH kommt mit einem Installationsprogramm daher. Nach der Installation generieren wir für die bestehenden Gruppen das group-File:

```
mkgroup -l >> ..\etc\group
```

Nun brauchen wir noch das passwd-File. Dieses erstellen wir mit folgendem Befehl:

```
mkpasswd -k -u marco >> ..\etc\passwd
```

Diese beiden Files werden im Verzeichnis etc (von OpenSSH) gespeichert. Dort liegt auch das Konfigurationsfile sshd_config für den SSH-Server. In diesem nehmen wir noch ein paar Änderungen vor:

```
StrictModes no


RSAAuthentication no

PubkeyAuthentication yes

AuthorizedKeysFile     /etc/authorized_keys


GatewayPorts yes
```

Nun können wir den SSH-Server starten. Er erscheint in den Services von Windows unter dem Namen OpenSSH Server.
Danach können wir uns mit PuTTY auf diesem Server anmelden.

3.1 Anmeldung mit Schlüsselpaar

Um nicht bei jedem Login das Passwort eingeben zu müssen, erstellen wir ein Schlüsselpaar. Auf dem Server können wir das mit folgendem Befehl tun:

```
ssh-keygen -f marco.key -t rsa
```

Die soeben erstellte Datei marco.key.pub verschieben wir in das Verzeichnis etc und geben ihm den neuen Namen authorized_keys. Die Datei marco.key ist der private Schlüssel zu dem wir Sorge tragen müssen.
Diesen privaten Schlüssel können wir mit dem PuTTY Key Generator in einen privaten Schlüssel für PuTTY konvertieren. Dazu im PuTTY Key Generator Conversions - Import Key wählen und die Datei marco.key öffnen. Danach den Schlüssel durch Save private key speichern. In PuTTY kann dieser Schlüssel unter Connection – SSH – Auth angegeben werden.

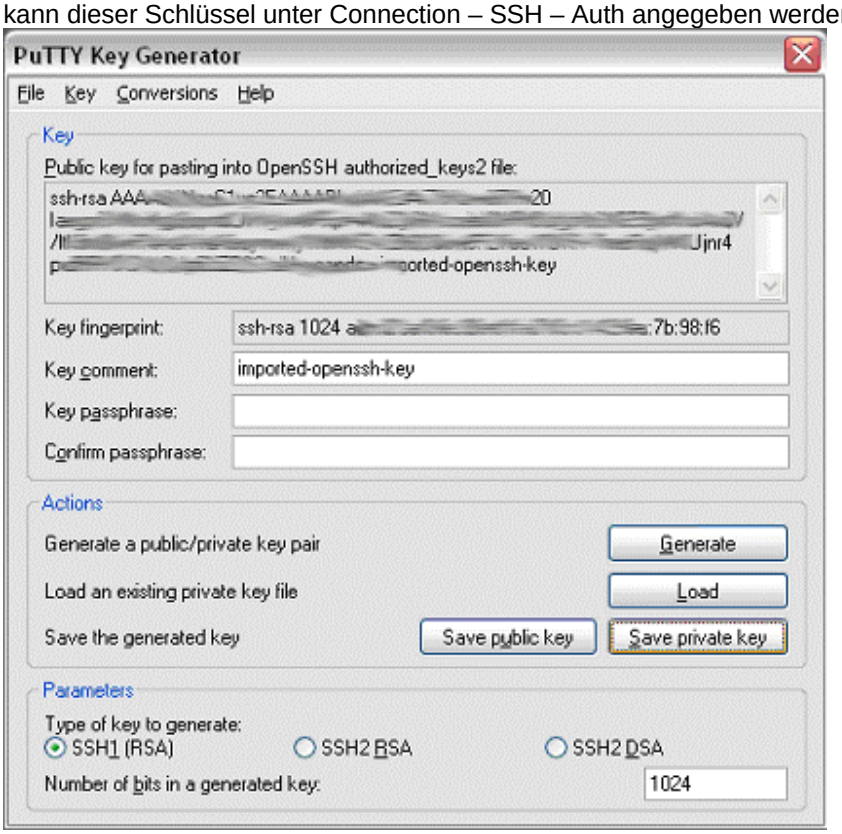

Abbildung 1: PuTTY Key Generator

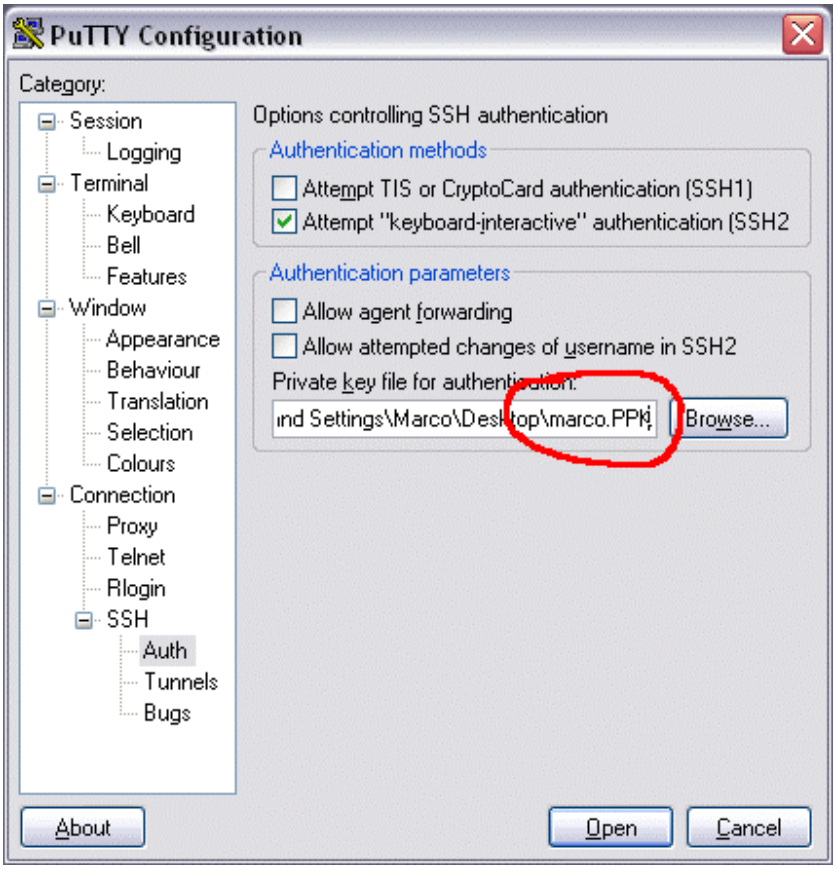

Abbildung 2: PuTTY mit Key-Auth

4 SSH-Tunnel

Die Kommunikation zwischen zwei Computern im Internet geht jeweils über Ports. Ein Webserver läuft in der Regel auf Port 80 (http). Weitere verbreitete Ports sind 110 (pop3), 25 (smtp), 20 (ftp-data) und 21 (ftp). RemoteDesktop läuft auf dem Port 3389. Eine Firewall kann diese Ports blocken. Um trotzdem auf so einen Service zuzugreifen, kann dies durch einen SSH-Tunnel erreicht werden. Sofern die Firewall den Port für SSH durchlässt (Port 22). Ein weiterer Vorteil eines SSH-Tunnels ist die verschlüsselte Kommunikation. Dabei wird z.B. bei einem Tunnel für pop3 das Passwort nicht mehr unverschlüsselt übertragen.
Ein SSH-Tunnel für eine FTP-Verbindung ist nicht möglich, da dies das FTP-Protokoll nicht zulässt. Der FTP-Server baut nämlich eine Rückwärtsverbindung zum Client auf, dabei ist der verwendete Port nicht vorher bekannt.

4.1 Funktionsweise

Ein Client-PC baut eine SSH-Verbindung zu einem Server auf. In den Parametern wird angegeben, welcher Port getunnelt werden soll. Die folgende Darstellung soll dies verdeutlichen:

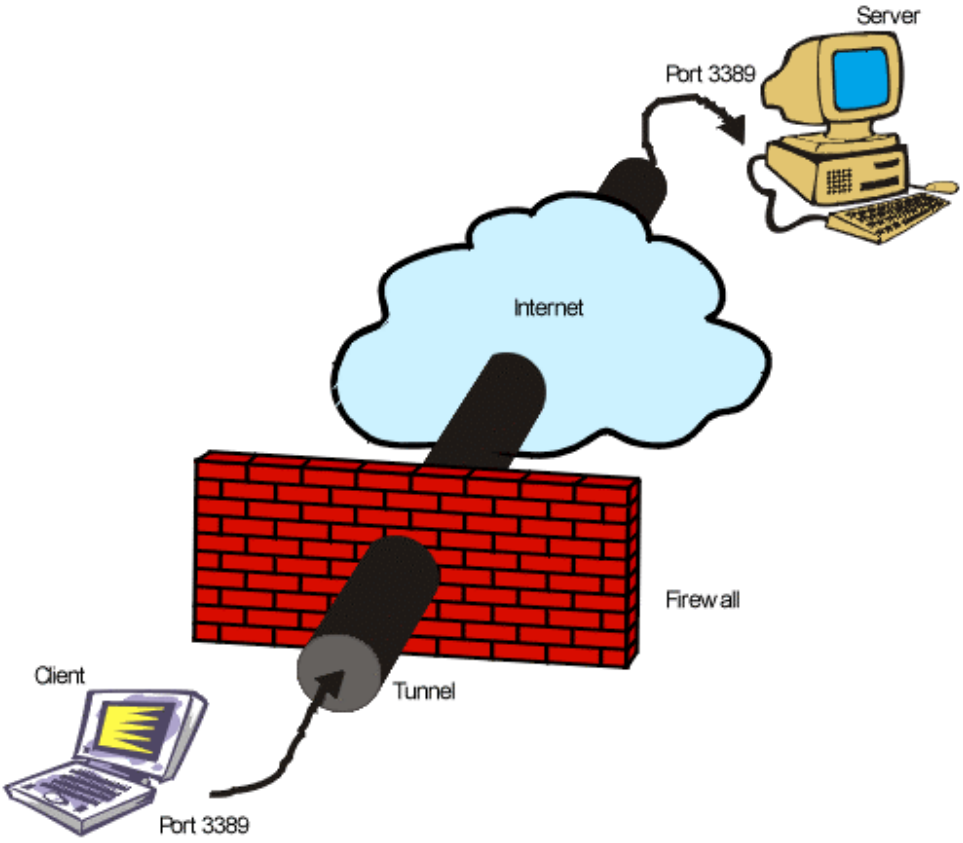

Abbildung 3: Tunnel

Nach dem Aufbau den Tunnels hört der Client auf dem Port 3389 und sendet alle Pakete durch den Tunnel auf den Server. Der Server nimmt die Pakete entgegen und sendet sie an den Port 3389. Der Tunnel selbst läuft auf dem Port 22. Die Firewall muss also nur den Port 22 durchlassen.

4.2 Remote-Desktop durch einen SSH-Tunnel

4.2.1 Problemstellung

Wir möchten mit dem Notebook von der Schule per Remote-Desktop auf den PC zu Hause zugreifen. Die Firewall der FHSO blockiert diesen Port.

4.2.2 Lösung

Den OpenSSH-Server haben wir auf dem Heim-PC bereits installiert, ebenso muss natürlich Remote-Desktop laufen. Mit dem Notebook bauen wir zuerst einen SSH-Tunnel auf. Er soll lokal auf dem Port 3389 hören und alles durch den Tunnel schicken. Am Ende des Tunnels sollen die Pakete wieder an den Port 3389 geschickt werden. Dafür erstellen wir in PuTTY eine neue Session. Bei Host Name tragen wir die IP-Adresse des Heim-PCs ein, als Protokoll wählen wir SSH. Unter Connection – SSH – Tunnels fügen wir einen neuen Tunnel ein. Source port ist 3389, Destination ist 127.0.0.1:3389.

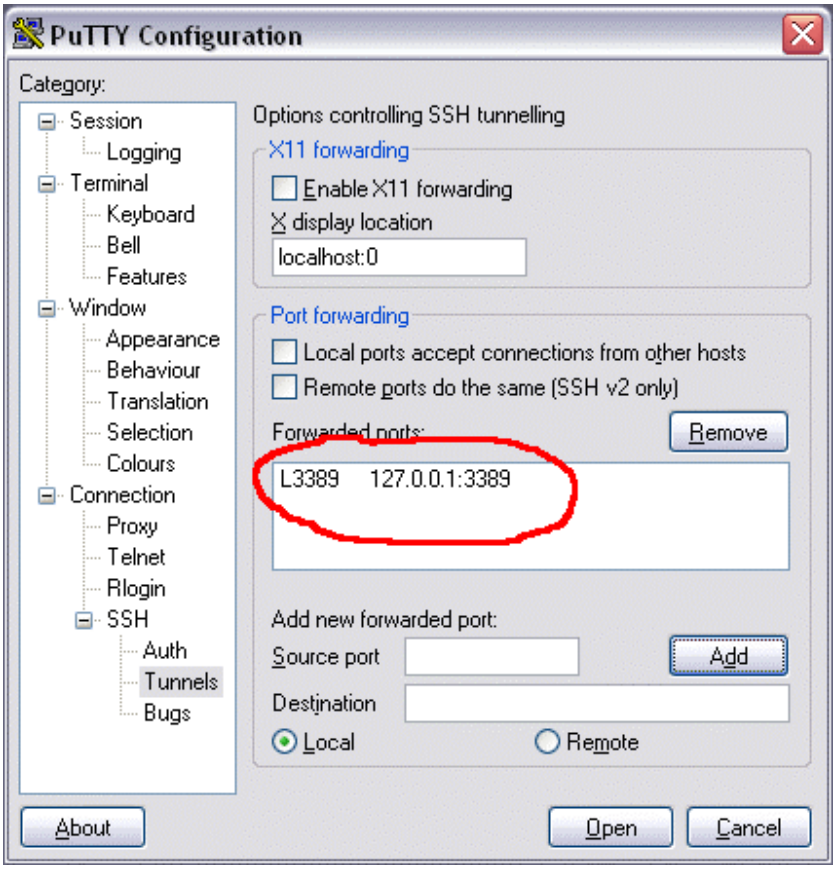

Abbildung 4: Tunnel für Remote Desktop

Sind alle Einstellungen gemacht, können wir die Verbindung aufbauen und uns zu Hause auf dem PC authentifizieren. Danach steht der Tunnel. Jetzt können wir uns per Remote-Desktop durch den Tunnel hindurch anmelden. Bei der Remote Desktop Connection geben wir als Zielcomputer 127.0.0.1 an.

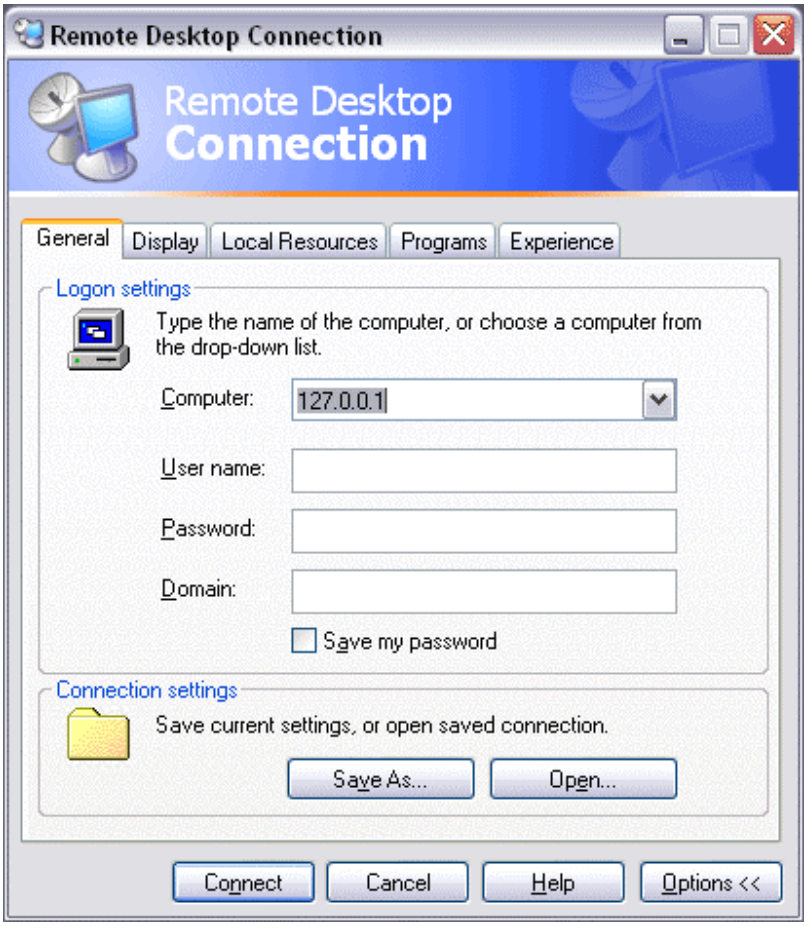

Abbildung 5: Remote Desktop Login

Problem bei Windows XP

Der Remote Desktop Client von Windows XP lässt keine Verbindung auf den Localhost (127.0.0.1) zu. Dem kann man folgendermassen abhelfen: Kopieren der beiden Dateien mstsc.exe und mstscax.dll aus dem Windows\System32 Verzeichnis in ein beliebiges anderes Verzeichnis. Dann bei den Eigenschaften von mstsc.exe den Compatibility mode auf Windows 98 / Windows Me setzen. Danach muss natürlich sichergestellt werden, dass sich die Konfigurationsdatei von Remote-Desktop (Endung rdp) mit der „neuen“ mstsc.exe öffnet.

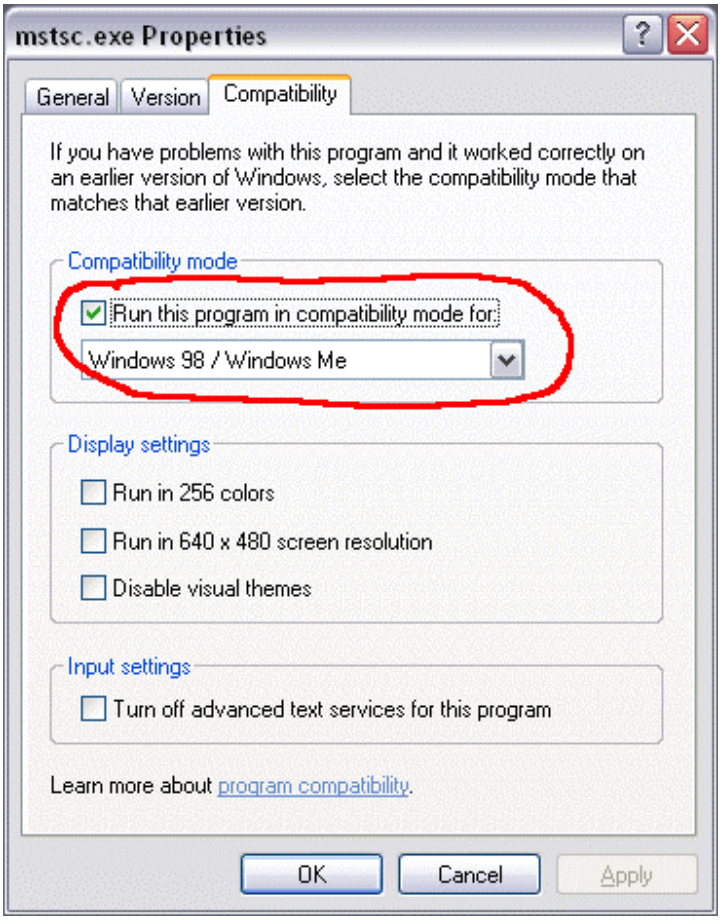

Abbildung 6: Kompatibilitätsmodus

4.3 Reverse Tunnel

4.3.1 Problemstellung

Wir möchten in der Schule auf dem Notebook einen Webservice entwickeln. Zum Testen muss dieser vom Internet aus erreichbar sein. Der Webservice läuft auf dem Webserver auf Port 80. Konkret angewendet habe ich dieses Szenario beim Entwickeln des Webservice für das CodeDuel bei codezone.ch (http://www.00001001.ch/CodeDuel/)

4.3.2 Lösung

Wir brauchen dazu wieder den Computer zu Hause. Er soll auf die externen http-Anfragen hören und diese durch den Tunnel auf das Notebook schicken.

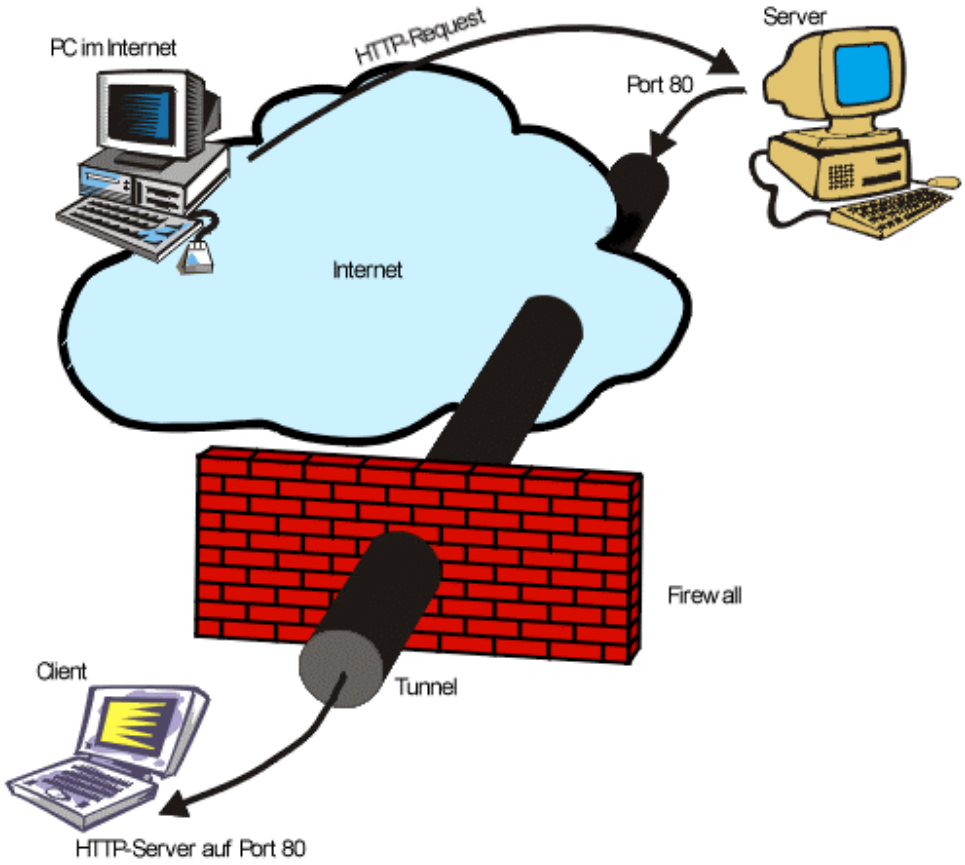

Abbildung 7: Reverse Tunnel

Mit PuTTY bauen wir einen Remote-Tunnel auf. Source port 80, Destination 127.0.0.1:80, Remote.

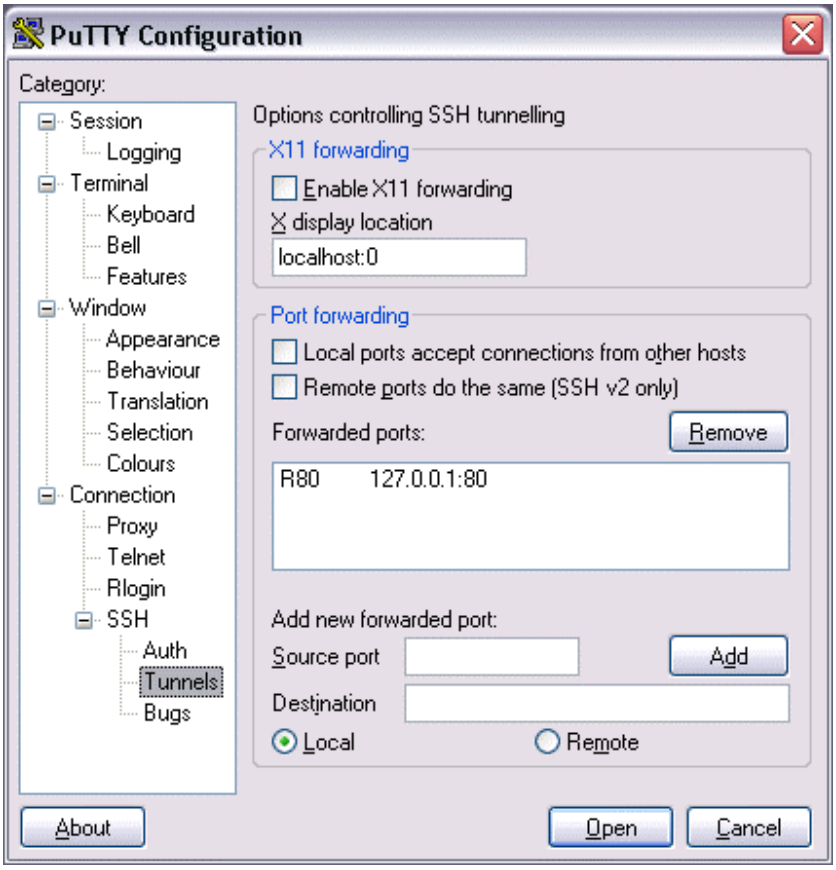

Abbildung 8: Konfiguration Reverse-Tunnel

Um die Einstellungen zu testen, braucht es noch einen dritten Computer. Für das konkret geschilderte Problem ist dies dann der CodeZone-Server. Wir können die Konfiguration mit der Lunchbox testen. Einloggen auf die Lunchbox und folgendes Kommando eingeben:

```
lynx 217.162.245.222/web/test.html
```

217.162.245.222 ist dabei die IP-Adresse des Computers zu Hause. Wenn dieser hinter einer Firewall steht, muss der Port 80 natürlich geforwardet werden. /web/test.html ist eine Test-HTML-Seite auf dem Notebook.

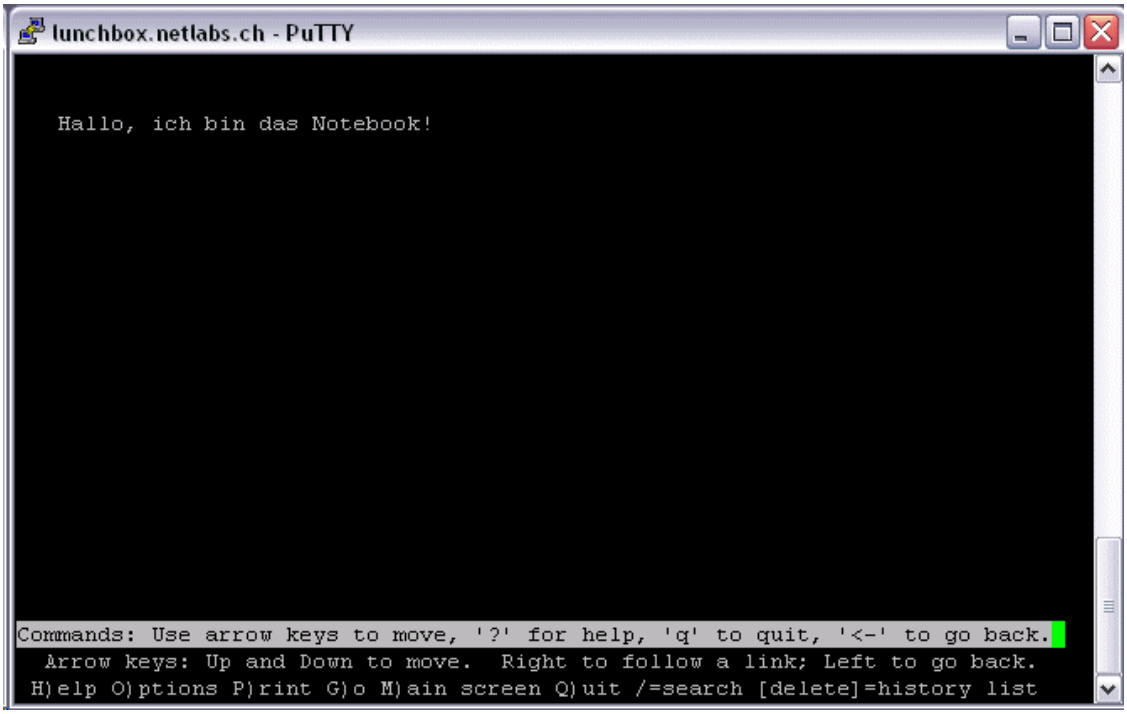

Abbildung 9: Reverse-Tunnel Test

4.3.3 TCP-Dump mit Ethereal

Da die Kommunikation durch einen SSH-Tunnel stets verschlüsselt ist, sieht man mit Ethereal nicht viel interessantes. Folgende beiden Screenshots zeigen aber den Unterschied zwischen dem Download der HTML-Seite durch den Tunnel und dem direkten Download ohne Tunnel.

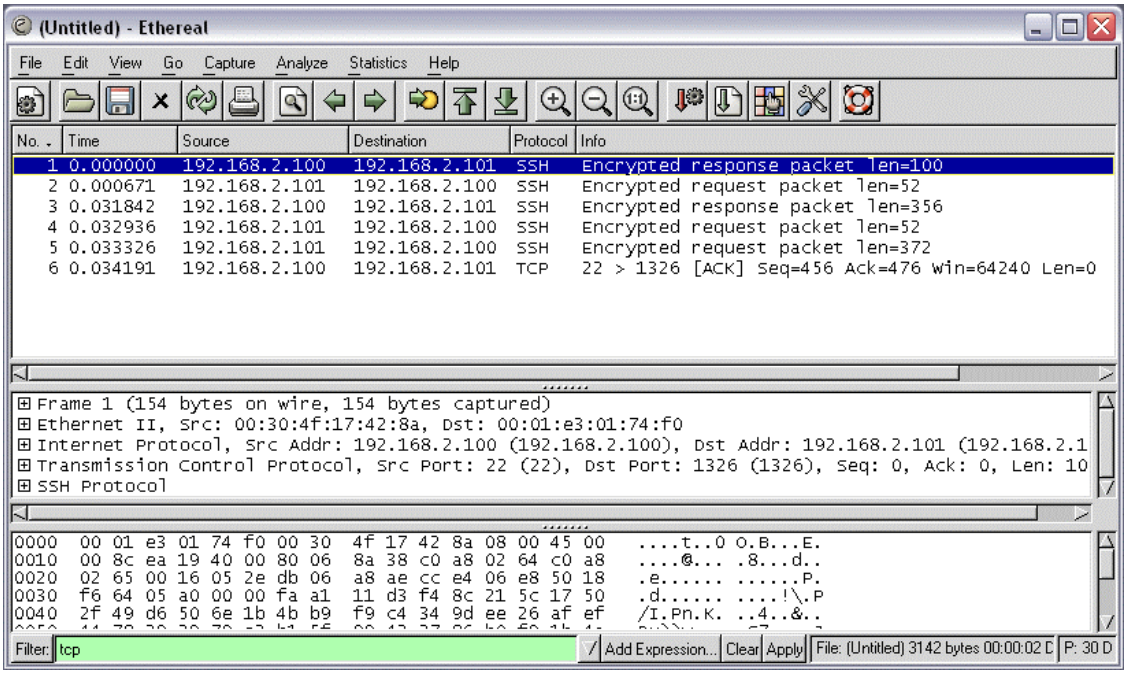

Abbildung 10: TCP-Dump mit Tunnel

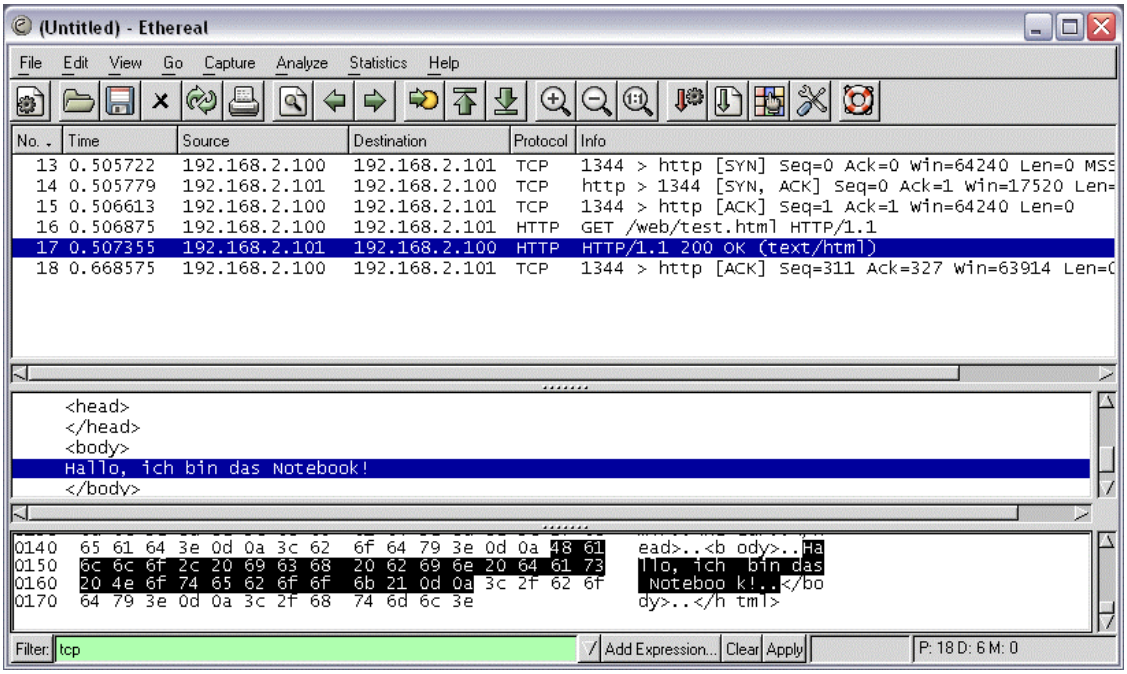

Abbildung 11: TCP-Dump ohne Tunnel

Anmerkung: Für diesen Dump war natürlich das Notebook im gleichen Netz wie der Computer, der die HTML-Seite angefordert hat.

4.4 Weitere Anwendungsbeispiele

SSH-Tunnels sind wirklich sehr einfach und genial. Ein Beispiel dafür ist TetriNET. TetriNET läuft auf dem Port 31457, welcher von der FHSO-Firewall auch geblockt wird. Dazu wird lediglich ein Tunnel zum SSH-Server im Internet (Lunchbox?!) aufgebaut, der lokal auf dem Port 31457 hört, die Pakete durch den Tunnel schickt und am Ende des Tunnels die Pakete auf den TetriNET Server weiterleitet. In TetriNET muss dann natürlich bei den Client Settings Localhost eingetragen werden.

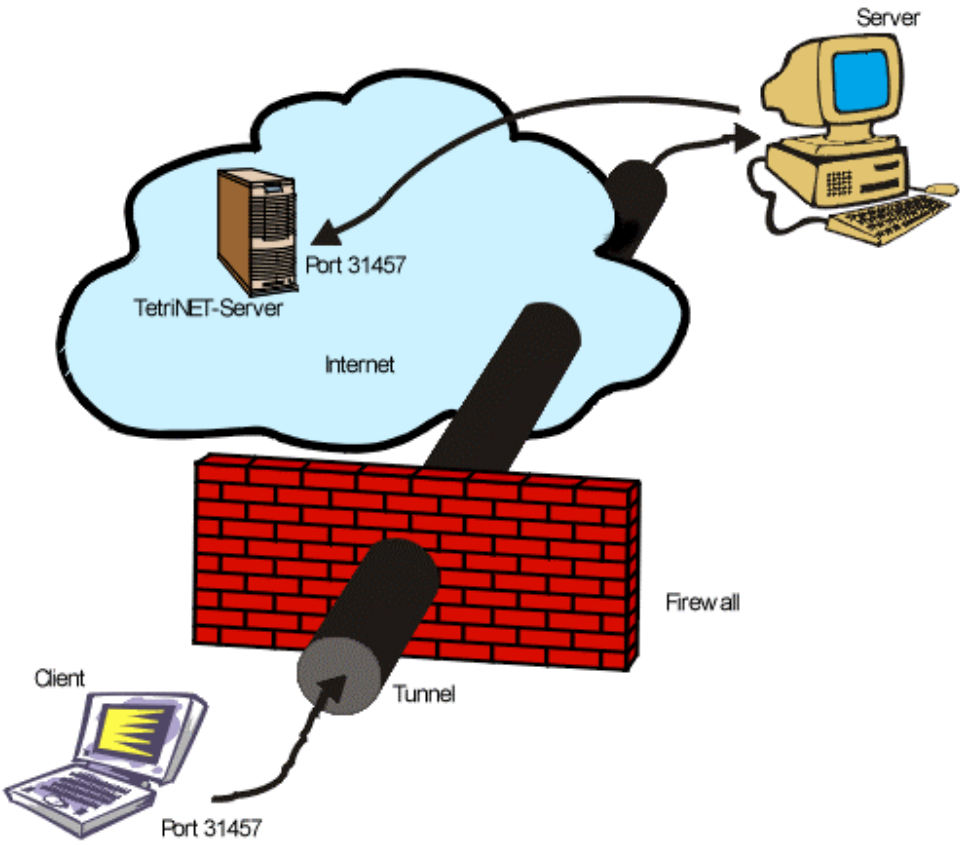

Abbildung 12: TetriNET

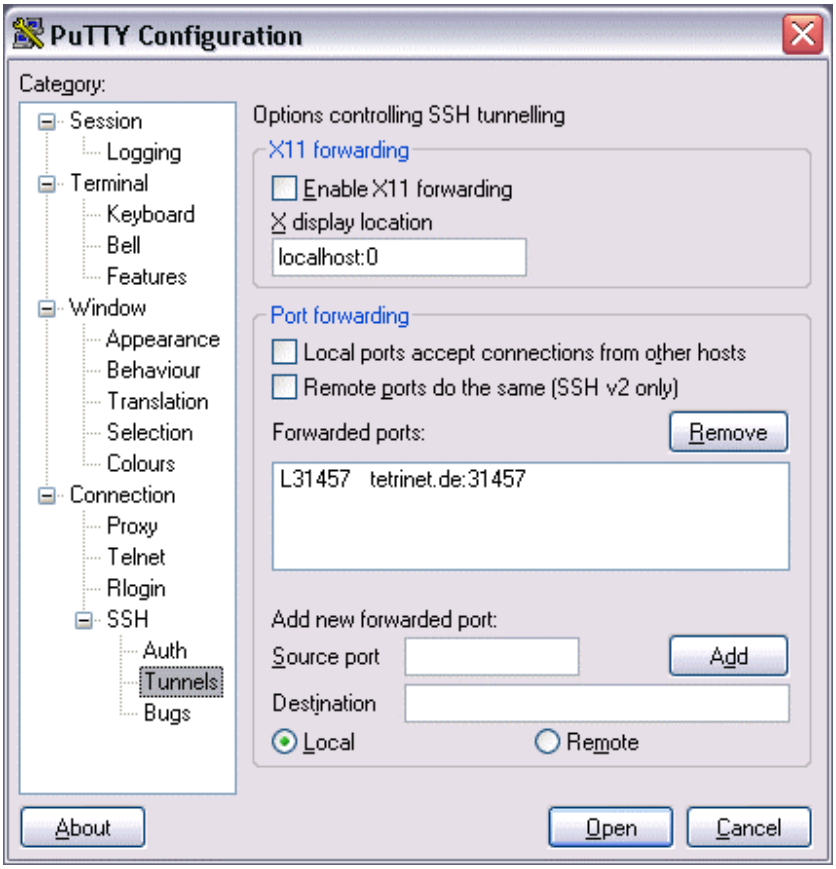

Abbildung 13: Einstellungen TetriNET

Mit einem kleinen Hack können sogar auch die Spiele auf Brettspielwelt.de gespielt werden. Diese Spiele laufen in einem Java-Applet. Die HTML-Seite, welche das Applet enthält, ist lokal zu speichern, damit man den Applet-Parameter für die Server-IP auf Localhost ändern kann.