

## SSH – Secure Shell



Semesterarbeit Michael Vogt, HFT-SO 2008  
michu@neophob.com

## INHALTSVERZEICHNIS

Inhaltsverzeichnis.....	2
Einleitung.....	3
Grundlagen.....	3
Übersicht.....	3
SSH Layer.....	5
Verwendete Server Software.....	6
SShd Konfiguration.....	6
Verwendete Client Software.....	7
PuTTY.....	7
WinSCP.....	8
Die 3 SSH Layer.....	9
SSH Transport Layer.....	9
Server Authentisierung.....	10
SSH Authentication Layer - Authentifizierungsmethoden.....	11
Passwort.....	11
Public Key.....	11
Hostbased.....	12
Weitere Methoden.....	12
SSH Connection Layer – Bitte um Verbindung.....	13
SSH in Action – Praktischer Einsatz.....	14
Authentifizierung mit einem Public Key.....	14
Die Public Key Authentifizierung Windows-to-Linux:.....	14
Die Public Key Authentifizierung Linux-to-Linux:.....	16
Agent forwarding.....	17
Hostbased Authentifizierung.....	18
Lokaler Tunnel.....	20
RDP over SSH.....	20
SMB over SSH (net use).....	21
Dynamic Tunnel.....	23
DNS Leak.....	24
Remote Tunnel - out from the inside.....	26
Remote RDP Tunnel.....	26
Windows SSH Server mit Cygwin.....	28
Bypass Proxy-Server.....	29
Proxy Authentifizierungen.....	29
Connect-NTLM Proxy.....	31
NTLMaps.....	33
Bypass Firewalls.....	34
HTTPTunnel.....	35
DNS Tunnel (IP-over-DNS).....	37
ICMP Tunnel.....	38
Under Attack – (Anti) SSH Tools.....	40
DSniff v2.4.....	40
GuessWho v0.44:.....	41
SSHatter v0.6.....	41
THC Hydra v5.4.....	42
ScanSSH v2.0.....	42
SSH-Privkey-Crack.....	43
PuTTY Private Key Crack.....	44
SSHDFilter v1.5.4.....	44
Weitere Tools im Zusammenhang mit SSH.....	45
Port-Knocking.....	45
SSH Best Practices.....	46
Glossar.....	47
Referenzen.....	48
Bildreferenzen.....	48

## EINLEITUNG

Diese Semesterarbeit soll den praktischen Einsatz des SSH Protokolls im Alltag aufzeigen, keinesfalls möchte ich mich in kryptografischen Details verlieren. Ein gewisses Mass an Theorie ist natürlich notwendig, jedoch möchte ich diese Arbeit primär als praktisches Nachschlagewerk verwenden.

Ich gehe bei meinen Anleitungen davon aus, dass als Client OS Windows 2k/XP/Vista verwendet wird und als Server ein GNU Linux System. Das ist jedoch keine Anforderung, da die meisten verwendeten Tools Cross Plattform tauglich sind.

## GRUNDLAGEN

### ÜBERSICHT

Der geschichtliche Hintergrund von SSH und eine Funktions-Übersicht ist auf Wikipedia<sup>1</sup> meiner Meinung nach gut beschrieben:

*Secure Shell oder SSH ist sowohl ein Programm als auch ein Netzwerkprotokoll, mit dessen Hilfe man sich über eine verschlüsselte Netzwerkverbindung auf einem entfernten Computer einloggen und dort Programme ausführen kann. Die neuere Version SSH2 bietet weitere Funktionen wie Datenübertragung per SFTP.*

*Die IANA hat dem Protokoll den TCP-Port 22 zugeordnet.*

#### **Geschichte**

*Die erste Version des Protokolls (jetzt SSH-1 genannt) wurde 1995 von Tatu Ylönen als Reaktion auf eine Passwort-Sniffing-Attacke entwickelt. Er veröffentlichte seine Implementation 1995 als Freeware, die daraufhin relativ schnell an Popularität gewann; Ende des Jahres 1995 zählte man bereits 20.000 Benutzer in fünfzig Ländern.*

*Im Dezember gründete Tatu Ylönen die Firma SSH Communications Security, um SSH zu vermarkten und weiterzuentwickeln. Die Originalsoftware enthielt ursprünglich Open-Source-Quellcode, entwickelte sich aber im Laufe der Zeit immer mehr zu proprietärer Software.*

*Nachdem einige Schwachstellen in der Integritätsprüfung von SSH-1 bekannt geworden waren, wurde 1996 mit SSH-2 eine überarbeitete Version des Protokolls entwickelt. Sie ist inkompatibel zu SSH-1. Dabei wurde unter anderem das Protokoll in verschiedene Einzelteile aufgegliedert und somit die Verwendung sicherer Verschlüsselungs- und Authentifikations-Algorithmen ermöglicht. Damit wurde die Schwachstelle beseitigt, und derzeit gilt das Protokoll als sicher.*

*1999 wurde der Wunsch nach einer freien Implementation von SSH laut, und aus der letzten freien Version der Originalimplementation entwickelte sich das separate OpenSSH-Projekt. Spätestens seit dieser Zeit existiert das SSH-Protokoll in zwei Implementationen: Als Open-Source-Software (OpenSSH) und als proprietäre Software (Produktname: SSHTectia), entwickelt und vertrieben von der Firma SSH Communications Security, also den Original-Entwicklern rund um Ylönen.*

<sup>1</sup> [http://de.wikipedia.org/wiki/Secure\\_Shell](http://de.wikipedia.org/wiki/Secure_Shell)

2005, also zehn Jahre nach der Original-Entwicklung, ist die Firma SSH Communications Security mit der Generation 3 (SSH G3) an die Öffentlichkeit gegangen. Dieses Protokoll unterstützt die Verwendung des proprietären Snakeoil<sup>2</sup>-Algorithmus „CryptiCore“ (Vorteile angeblich vor allem im Datendurchsatz, dies ist in der Praxis aber völlig irrelevant, da SSH auf heutigen PCs in Leitungsgeschwindigkeit funktioniert), der jedoch als unsicher gilt (siehe „Verschlüsselung“). Die anderen, etablierten Verschlüsselungsalgorithmen, werden weiterhin ebenfalls unterstützt. 2006 wurde dieses Protokoll (Version 2) von der IETF als Internetstandard vorgeschlagen. Eine Zertifizierung nach FIPS-Standard 140-2 (FIPS = Federal Information Processing Standard) besteht bereits länger.

Wichtiger als der neue Algorithmus ist die erstmalige Unterstützung des SSH-Protokolls auf dem Mainframe (IBM z/OS). Damit existiert nun nach Hersteller-Angaben eine durchgängige Unterstützung der Plattformen Unix-Derivate (BSD, Linux, Solaris, AIX, HP-UX, u.a.), Windows, IBM z/OS.

### **Verwendung**

SSH ermöglicht eine sichere, authentifizierte und verschlüsselte Verbindung zwischen zwei Rechnern über ein unsicheres Netzwerk. Dadurch dient es unter anderem als Ersatz für die Vorgänger rlogin, telnet und rsh; diese übertragen jeglichen Netzverkehr, darunter auch die Passwörter, unverschlüsselt und sollten daher nicht mehr verwendet werden.

Das ursprüngliche Anwendungsgebiet ist das Anmelden an entfernten Rechnern über ein Netzwerk (meistens das Internet), doch insbesondere SSH-2 ist nicht nur auf Terminalfunktionen beschränkt.

- \* SFTP und SCP bieten kryptographisch sichere Alternativen zu FTP und rcp.
- \* X11 kann über SSH transportiert und somit gesichert werden
- \* Über SSH können beliebige TCP/IP-Verbindungen getunnelt werden (Portweiterleitung); dabei wird jeweils ein einzelner Port von einem entfernten Server auf den Client weitergeleitet oder umgekehrt. So kann etwa eine ansonsten unverschlüsselte VNC-Verbindung abgesichert werden.
- \* Ein SSH-Client kann sich wie ein SOCKS-Server verhalten und ermöglicht somit einen automatisierten Zugriff auf entfernte Rechner durch den SSH-Tunnel, etwa zum Umgehen einer Firewall.
- \* Über SSHFS kann ein entferntes Dateisystem auf dem lokalen Rechner gemountet werden.
- \* Wenn der Server A dem Server B ohne interaktive Eingabe des Passworts SSH-Zugriff gestatten soll, trägt man den öffentlichen Schlüssel von B in die SSH-Konfiguration von A ein (authorized\_keys). Dann kann B z. B. von der Kommandozeile via SSH auf A Kommandos ausführen oder mit SCP Backupkopien machen.
- \* Mit "ssh-keyscan" kann der öffentliche Schlüssel eines entfernten Rechners ausgelesen werden. Damit kann man z. B. feststellen, ob der Rechner, der sich derzeit hinter einem DynDns-Namen verbirgt, wirklich der eigene Server ist.

Wahlweise kann die Verbindung auch komprimiert werden, um die Datenübertragung zu beschleunigen und Bandbreite zu sparen.

---

<sup>2</sup> Schlangenöl (aus dem Englischen snake oil) ist (hauptsächlich im Bereich von Software) die Bezeichnung für ein Produkt, das wenig oder keine echte Funktion hat, aber als Wundermittel zur Lösung vieler Probleme vermarktet wird.

Damit lassen sich nun grundsätzlich **drei Anwendungsszenarien** darstellen:

- \* *Secure System Administration (Sichere Systemverwaltung)*  
zur Absicherung der Fernverwaltung von Servern. Ersetzt Telnet, Rlogin etc.
- \* *Secure File Transfer (Sicherer Dateitransfer)*  
zur Herstellung sicherer, automatisierter und interaktiver Dateitransfers.
- \* *Secure Application Tunneling (Sicheres Tunneln)*  
zum transparenten Schutz TCP/IP-basierender Anwendungen als „End-to-End-Security“.

Anmerkung: Natürlich kann SSH auch über mehrere Stationen laufen.

## SSH LAYER

Eine erste Übersicht über den Aufbau des SSH Protokolls:

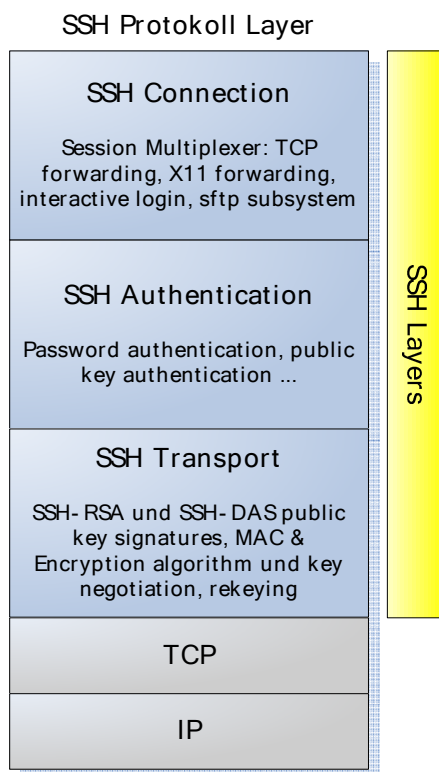


Abbildung 1: SSH Subsystems

Das SSH Protokoll (rfc: #4251) ist modular aufgebaut und besteht aus 3 Subsystemen:

SSH Transport (rfc: #4253):

Dieser Layer ist für die sichere Kommunikation während und nach der Authentifizierung zuständig. Konkret bedeutet dies Verschlüsselung und Entschlüsselung der Daten, Authentifizierung des Servers (ist der Server derjenige, den er vorgibt zu sein, Stichwort: Man-in-the-middle attack) und sicherstellen der Datenintegrität mit Hilfe des Message Authentication Code (MAC) Algorithmus. Weiter handelt dieser Layer die Daten-Komprimierung um die Verbindungsgeschwindigkeit zu erhöhen.

SSH Authentication (rfc: #4252):

Wurde eine sichere Verbindung hergestellt, gilt es den User zu identifizieren. Ein User kann sich auf verschiedene Arten an einem SSH Server anmelden, z.B. via Public Key oder Passwort-Eingabe.

SSH Connection (rfc: #4254):

Wurde der User authentisiert, wird die Verbindung mittels eines Multiplexers in mehrere Channels aufgesplittet. Ein Channel dient z.B. als Terminal, ein anderer forwarded einen TCP Port. Die Channels können dynamisch geöffnet und geschlossen werden, ohne dass die Verbindung unterbrochen wird.

## VERWENDETE SERVER SOFTWARE

Als SSH Daemon verwende ich den Standard SSH Daemon, welcher bei meiner Distribution per default installiert ist. In meinem Fall (und das wird bei den meisten x86/x64 Distributionen der Fall sein) ist dies der OpenSSH Daemon:



Abbildung 2: OpenSSH Logo

Zur Zeit ist Version 4.7 von OpenSSH aktuell. Kleinere, d.h. vor allem auf Embedded Linux spezialisierte Distributionen wie z.B. OpenWRT<sup>3</sup> verwenden einen „abgespeckten“ SSH Daemon wie Dropbear<sup>4</sup>.

## SSHD Konfiguration

Ausschnitt aus dem SSHd Konfigurationsfile /etc/ssh/sshd\_config:

```
# SSHd kann auf mehreren Port „hören“
Port 22
Port 443

# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#Privilege Separation is turned on for security
#(nicht der ganze daemon wird als root ausgeführt, sondern nur das nötigste)
UsePrivilegeSeparation yes

# Authentication:
LoginGraceTime 600
PermitRootLogin yes
# The option StrictModes specifies whether ssh should check user's permissions in their home
# directory and rhosts files before accepting login. This option must always be set to yes
# because sometimes users may accidentally leave their directory or files world-writable.
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes
```

<sup>3</sup> <http://openwrt.org/>

<sup>4</sup> <http://matt.ucc.asn.au/dropbear/dropbear.html>

## VERWENDETE CLIENT SOFTWARE

Unter Windows gibt es ein sehr gutes Open Source (MIT Lizenz) Produkt im Zusammenhang mit SSH – PuTTY<sup>5</sup>. Ein weiterer freier (Apache Lizenz) SSH Client ist Pedrosa<sup>6</sup>, dieser wurde mit DOT NET geschrieben und braucht eine Installation. Deswegen verwende ich ausschliesslich PuTTY – PuTTY ist ein EXE File, dass keine Installation benötigt.

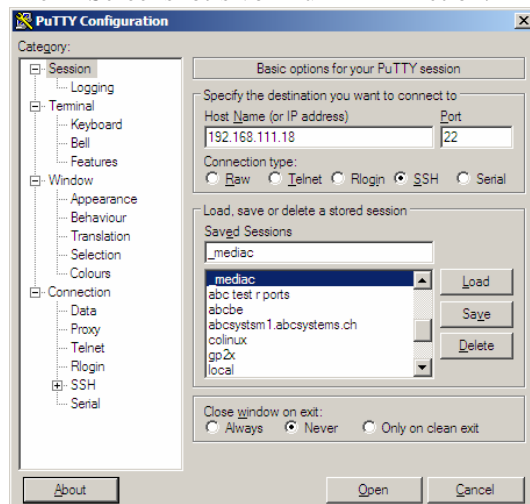
## PuTTY

PuTTY besteht aus folgenden Komponenten:

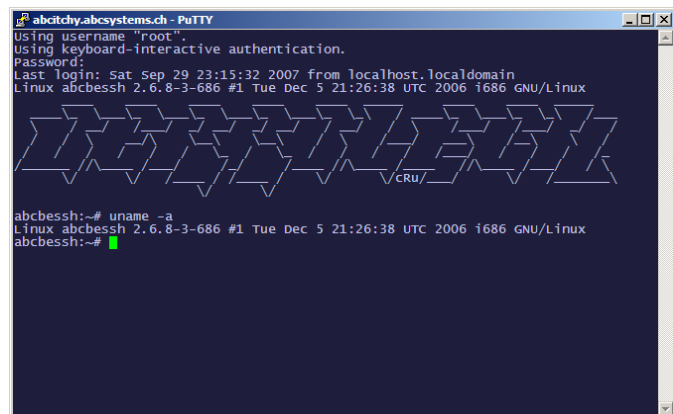
- PAGEANT.EXE  
PuTTY authentication agent. Ein System-Tray Tool, welches public key's „verwaltet“.
- PSCP.EXE  
PuTTY Secure Copy Client, Console Version
- PSFTP.EXE  
PuTTY Secure File Transfer Client, Console Version
- PUTTY.EXE  
Der SSH Client, GUI Version
- PLINK.EXE  
Der SSH Client, Console Version
- PUTTYGEN.EXE  
PuTTY key generator, erstellt private und public SSH key's (Achtung, diese Key Files sind nicht mit den OpenSSH Key Files kompatibel)

PuTTY bietet die Möglichkeit, eine Session mit allen relevanten Einstellungen (Ziel Host und Port, Proxy, Tunnel...) zu sichern. Auf diese gespeicherten Sessions kann auch von den Console Tools, via Sessionname, zugegriffen werden.

Hier 2 Screenshot's von PuTTY in Action:



**Abbildung 3: PuTTY Konfiguration**



**Abbildung 4: PuTTY in Action**

<sup>5</sup> <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

<sup>6</sup> <http://en.poderosa.org/>

## WinSCP

Um Daten sicher über ein Netzwerk zu kopieren, verwende ich einen etwas komfortableren SFTP/SCP Client als PuTTY bietet, ich verwende WinSCP<sup>7</sup>. WinSCP verwendet die SSH Code-Basis von PuTTY. Die Beschreibung von der WinSCP Web Seite:

*WinSCP is an open source free SFTP client and FTP client for Windows. Legacy SCP protocol is also supported. Its main function is safe copying of files between a local and a remote computer.*

Ebenfalls auf der Web Seite von WinSCP befindet sich eine gute Beschreibung der SFTP und SCP Protokolle:

### *SFTP (SSH File Transfer Protocol)*

*Despite SSH in its name, it is designed to work over any reliable data stream, but WinSCP supports only operation over SSH, which is also by far its most common usage.*

*Being operated over SSH, it is secure protocol. In its basic functionality it is similar to old FTP, while having better designed advanced functionality. Unfortunately not all SFTP server implementation takes advantage of the advanced features, yet.*

*Especially in its later versions (from 4 upwards), it is more platform independent, compared to both FTP and SCP.*

*Unlike SCP, for connection with an SFTP server you do not need access to shell (although some implementations may still require that).*

### *SCP (Secure Copy Protocol)*

*SCP is mostly used with SSH-1. SCP is an older protocol but almost universally supported on Unix-like platforms as part of an SSH protocol suite. It is rarely supported on other platforms. SCP is a descendant of the ancient “rcp.”*

*The SCP protocol allows only file transfers in both directions. WinSCP is able to offer other features, implemented using common shell commands like cd, ls, pwd, rm, ln, etc. For this, WinSCP—unlike command-line SCP clients (which allow only file transfers)—requires full shell access and permission to execute other commands in addition to scp (see requirements). For access to non-UNIX servers it is necessary that the server have at least a UNIX-like shell.*

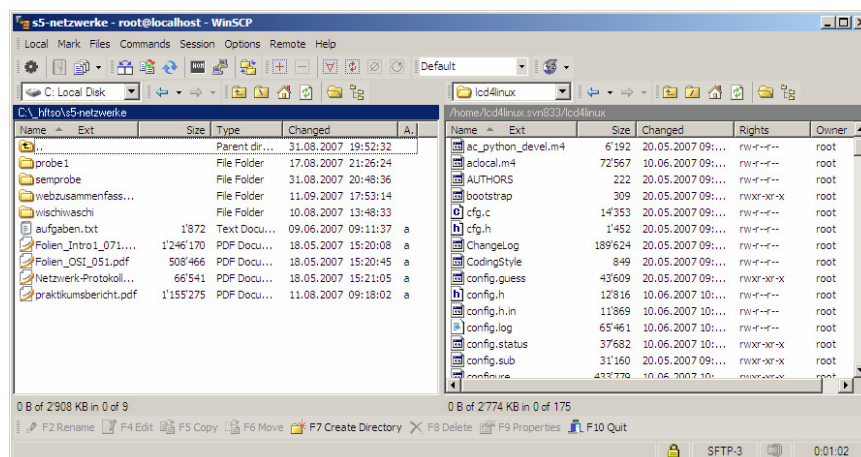


Abbildung 5: WinSCP in Action

<sup>7</sup> <http://winscp.net/>

## DIE 3 SSH LAYER

Das SSH Protokoll ist in 3 Layer aufgeteilt, welche unabhängig voneinander arbeiten.

### SSH TRANSPORT LAYER

Die Hauptaufgabe des Transport Layers ist der Aufbau einer sicheren und stabilen Verbindung zweier Hosts während und nach der Authentifizierung. Die Einleitung nach RFC4253<sup>8</sup>:

*The SSH transport layer is a secure, low level transport protocol. It provides strong encryption, cryptographic host authentication, and integrity protection.*

*Authentication in this protocol level is host-based; this protocol does not perform user authentication. A higher level protocol for user authentication can be designed on top of this protocol.*

*The protocol has been designed to be simple and flexible to allow parameter negotiation, and to minimize the number of round-trips. The key exchange method, public key algorithm, symmetric encryption algorithm, message authentication algorithm, and hash algorithm are all negotiated. It is expected that in most environments, only 2 round-trips will be needed for full key exchange, server authentication, service request, and acceptance notification of service request. The worst case is 3 round-trips.*

Folgende Aufgaben sind Bestandteil des Transport Layers:

- Key exchange (Kex)
- The public key algorithm to be used
- The symmetric encryption algorithm to be used
- The message authentication algorithm to be used
- The hash algorithm to be used

Das SSH Protokoll garantiert die Daten-Integrität (Sicherstellung, dass die Daten nicht verändert wurden) mit einem MAC - Message Authentication Code. SSHv1 verwendete noch Checksummen zur Sicherstellung der Daten-Integrität. Das hilft bei Übertragungsfehler, jedoch nicht bei böswillig manipulierten Daten, da ein Angreifer die Checksumme entsprechend korrigieren kann. Die Frage, ob eine Verschlüsselung Daten-Integrität garantiert, kann auch mit einem Nein beantwortet werden (aufgrund mangelndem kryptografischen Verständnis schenke ich dieser Aussage Glauben).

Vereinfacht kann man sagen, dass ein MAC einer kryptografischen Hash Funktion entspricht. Ein MAC besteht aus 2 Algorithmen (Signierung und Überprüfung). Die Integritäts-Überprüfung könnte auch mit dem Public Key Signaturverfahren garantiert werden, jedoch verhalten sich MAC in der Laufzeit besser (performanter).

Noch eine letzte Bemerkung betreffend MAC aus dem RFC4251:

*Because MACs use a 32-bit sequence number, they might start to leak information after 2\*\*32 packets have been sent. However, following the rekeying recommendations should prevent this attack. The transport protocol [SSH-TRANS] recommends rekeying after one gigabyte of data, and the smallest possible packet is 16 bytes. Therefore, rekeying SHOULD happen after 2\*\*28 packets at the very most.*

<sup>8</sup> <http://www.ietf.org/rfc/rfc4253.txt>

## Server Authentisierung

Jeder SSH Server wird mit dem eindeutigen SSH fingerprint identifiziert, welcher mit dem Tool ssh-keygen ausgelesen werden:

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
2048 93:f0:3b:3a:c4:0b:ee:f6:b2:47:e0:57:d9:64:0b:55 /etc/ssh/ssh_host_rsa_key.pub
```

Wird eine SSH Verbindung aufgebaut, speichert der SSH Client den fingerprint des SSH Servers lokal ab. Wird zu einem späteren Zeitpunkt wieder eine Verbindung mit diesem SSH Server aufgebaut, vergleicht der SSH Client den lokal gecachten fingerprint mit dem aktuellen Server fingerprint. Sind diese gleich ist alles in Ordnung; es ist der identische SSH Server. Stimmt der fingerprint des SSH Servers hingegen nicht mehr, zeigt der Client die entsprechende Warnung an:



**Abbildung 6: invalider SSH fingerprint**

Was ist passiert? Der fingerprint des SSH Servers wurde geändert. Das kann das Resultat eines Server-Updates sein (nach der Installation wird der Server-Key neu generiert) oder aber auch eine „man-in-the-middle“ Attacke.

Diese fingerprint Überprüfung hat jedoch eine Schwachstelle: der erste Verbindungsaufbau („initial trust problem“). Ist bereits eine „man-in-the-middle“ Attacke im lokalen Netzwerk aktiv, wird der „falsche“ SSH fingerprint in den Cache geschrieben – die Attacke wird nicht bemerkt.

## SSH AUTHENTICATION LAYER - AUTHENTIFIZIERUNGSMETHODEN

Ist die Verbindung zwischen Client und Server aufgebaut (Transport Layer), muss sich der Client beim Server authentifizieren. Der Server teilt dem Client die Authentifizierungsverfahren mit, welche er unterstützt, der Client wählt ein Verfahren aus und versucht sich zu authentifizieren.

Die Einleitung nach RFC4252<sup>9</sup>:

*The SSH authentication protocol is a general-purpose user authentication protocol. It is intended to be run over the SSH transport layer protocol [SSH-TRANS]. This protocol assumes that the underlying protocols provide integrity and confidentiality protection.*

Ein SSH Server bietet verschiedene Möglichkeiten an, um sich zu authentifizieren. Diese Methoden werden hier aufgeführt.

### Passwort

Die einfachste Art, sich an einem SSH Server anzumelden ist, dass der SSH Server nach einem User und dessen Systempasswort fragt.

Das Passwort wird im Authentication Layer unverschlüsselt übertragen – jedoch wird das Passwort durch den darunterliegenden SSH Transport Layer geschützt. Auszug aus RFC4252:

*Note that even though the cleartext password is transmitted in the packet, the entire packet is encrypted by the transport layer. Both the server and the client should check whether the underlying transport layer provides confidentiality (i.e., if encryption is being used). If no confidentiality is provided ("none" cipher), password authentication SHOULD be disabled. If there is no confidentiality or no MAC, password change SHOULD be disabled.*

Diese Variante wird kaum mehr verwendet, siehe Abschnitt PAM.

### Public Key

Diese Methode baut auf einem asymmetrischen Schlüsselpaar auf; der Public-Key und Private-Key. Der öffentliche Schlüssel wird auf allen Systemen hinterlegt, welche miteinander kommunizieren wollen. Der private Schlüssel bleibt auf dem Rechner des Benutzers und wird durch einen Passphrase (Passwort um den Schlüssel zu verwenden) geschützt.

Das Schema sollte den Public Key Vorgang verdeutlichen:

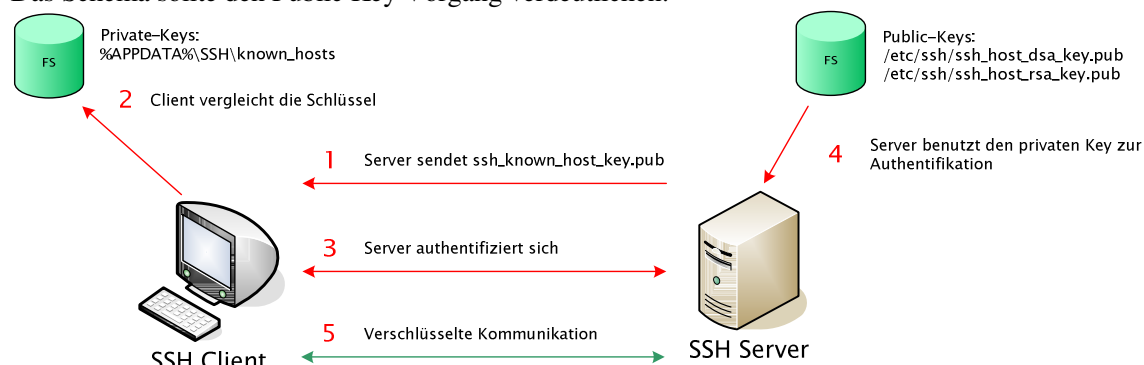


Abbildung 7: Public Key Vorgang

Der Private-Key wird also zu keiner Zeit übertragen. Siehe auch das Kapitel „Authentifizierung mit einem Public Key“.

<sup>9</sup> <http://www.ietf.org/rfc/rfc4252.txt>

## Hostbased

Diese Methode verwendet kein Username oder ein Passwort, sondern – Überraschung – den Hostnamen zur Authentifizierung. Dazu wird, ähnlich wie bei der Public Key Methode, ein Schlüsselpaar generiert. Diese Methode sollte sicherlich nicht in High-Security-Umgebungen verwendet werden, jedoch kann diese Methode z.B. für scriptgesteuerte Aktionen in einem mehr oder weniger sicheren Netz verwendet werden.

## Weitere Methoden

Es gibt weitere Authentifizierungs-Methoden, welche jedoch nicht im Detail angeschaut werden:

### PAM<sup>10</sup>:

Ähnlich wie die Password Methode, nur wird das Passwort nicht vom SSH Server überprüft sondern von der PAM Authentifizierungsschicht. Diese muss sowohl auf dem Server wie auch auf dem Client laufen. PAM ist die Standardauthentifizierung von Linux mittels der Datei /etc/passwd. PAM bietet Applikationen die Möglichkeit, sich via Standard API auf einem Server zu Authentifizieren.

### Kerberos/GSSAPI:

Eine Erweiterung des SSH Authentication Layers, definiert in RFC #4462. Erlaubt Authentifizierung via Key Distribution Server von Kerberos. Das Passwort liegt also nicht auf dem SSH Server. So sollte auch ein User-Account gegenüber einem Active Directory Server verifiziert werden.

### Smartcard:

Die Schlüssel werden auf einer Smartcard gespeichert. OpenSSH unterstützt Cyberflex Smartcards, die die PKCS#15 Infrastruktur unter OpenSC<sup>11</sup> unterstützen.

Dies ist keine abschliessende Liste von SSH Authentifizierungsmethoden, es gibt noch weitere Methoden wie RADIUS, S/Key OTP...

---

<sup>10</sup> PAM - Pluggable Authentication Modules are at the core of user authentication in any modern linux distribution.

<sup>11</sup> <http://www.opensc-project.org/>

## SSH CONNECTION LAYER – BITTE UM VERBINDUNG

Kommt der Connection Layer zur Anwendung, hat sich der Client bereits erfolgreich an einem SSH Server angemeldet und die Verbindung erfolgt verschlüsselt. Dieser Layer bietet dem User die offensichtlichste Funktion. Die Einleitung nach RFC4254<sup>12</sup>:

*The SSH Connection Protocol has been designed to run on top of the SSH transport layer and user authentication protocols ([SSH-TRANS] and [SSH-USERAUTH]). It provides interactive login sessions, remote execution of commands, forwarded TCP/IP connections, and forwarded X11 connections.*

*The 'service name' for this protocol is "ssh-connection".*

*This layer defines the concept of channels, channel requests and global requests using which SSH services are provided. A single SSH connection can host multiple channels simultaneously, each transferring data in both directions. Channel requests are used to relay out-of-band channel specific data, such as the changed size of a terminal window or the exit code of a server-side process. The SSH client requests a server-side port to be forwarded using a global request. Standard channel types include:*

- \* "shell" for terminal shells, SFTP and exec requests (including SCP transfers)*
- \* "direct-tcpip" for client-to-server forwarded connections*
- \* "forwarded-tcpip" for server-to-client forwarded connections*

Ein Shell Befehl kann ein Terminal Shell sein (z.B. BASH), ein Script das abläuft, SFTP/SCP secure Filetransfer oder auch ein anderes SSH Subsystem.

Wichtig für uns als Anwender ist die Tatsache, dass eine SSH Verbindung mehrere Connections haben kann. Eine Connection ist z.B. eine interaktive Shell, eine TCP-Port Weiterleitung etc. und kann während einer SSH Verbindung dynamisch geöffnet oder geschlossen werden.

Man kann also eine SSH Verbindung auch ohne interaktive Shell öffnen, z.B. um einen Remote Befehl abzusetzen oder TCP Ports weiterzuleiten.

Alle Connections werden gebündelt (multiplexed) und durch die darunterliegende Schicht gesichert.

---

<sup>12</sup> <http://www.ietf.org/rfc/rfc4254.txt>

## SSH IN ACTION – PRAKTISCHER EINSATZ

Wie schon in der Einleitung erwähnt, möchte ich diese Arbeit hauptsächlich als Nachschlagewerk verwenden, daher möchte ich hier Praxisnahe Beispiel-Anwendungen aufzeigen.

### AUTHENTIFIZIERUNG MIT EINEM PUBLIC KEY

Will man bei einer neuen SSH Verbindung nicht jedesmal User und Passwort eingeben, gibt es die Möglichkeit sich via Keypair (Private Key und Public Key) zu authentifizieren.

Eine Übersicht der Involvierten SSH Komponenten:

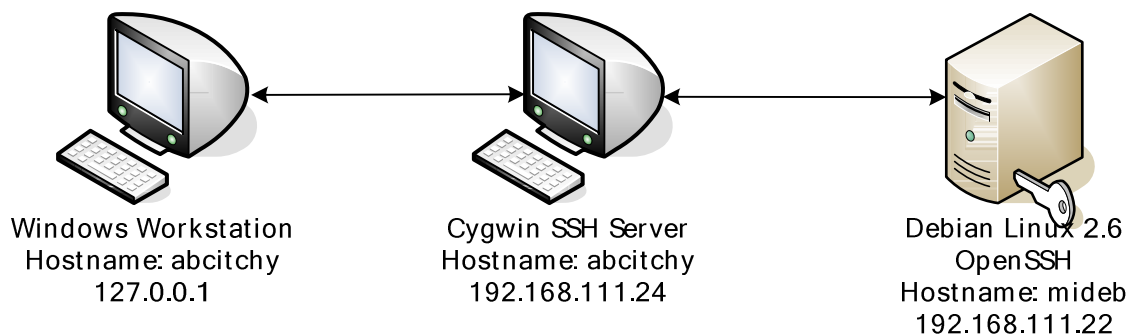


Abbildung 8: Übersicht der Laborumgebung

### Die Public Key Authentifizierung Windows-to-Linux:

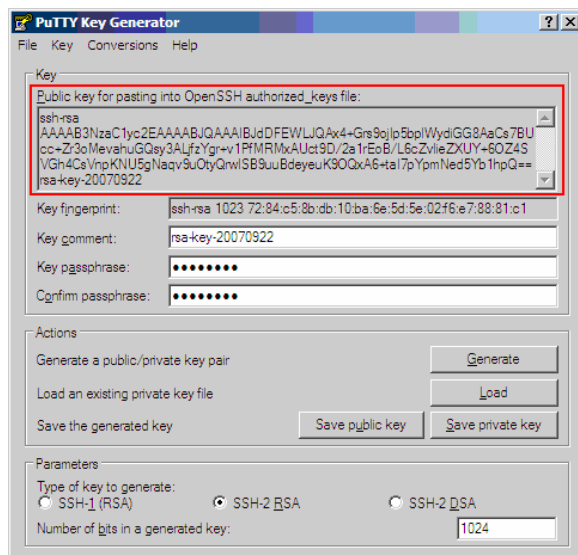


Abbildung 9: PuTTYGEN

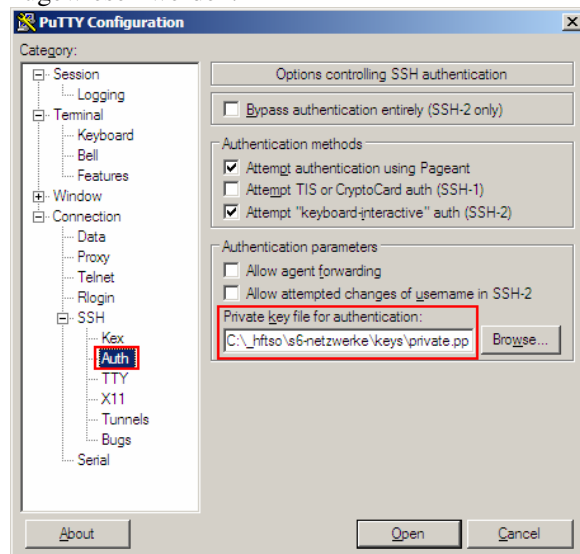
PuTTYGEN ist das Tool, welches PuTTY mitliefert um ein neues Schlüsselpaar zu generieren. Der private key sollte IMMER mit einem Passphrase geschützt werden. Wird der private key verwendet, muss zuerst dessen Passphrase eingegeben werden, der den privaten Schlüssel dechiffriert.

Der Private und der Public Key werden lokal abgespeichert (Save the generated key).

Wie kommt nun der Public Key des Users auf den SSH Server? Unter Windows muss der String „Public Key for pasting...“ (Rot markiert) in die Zwischenablage kopiert werden. Als nächster Schritt verbinden wir uns mit dem entsprechenden SSH Server und kopieren die Zwischenablage in das File `~\.ssh\authorized_keys2`. Auf dieses File sollte aus Sicherheitsgründen nur der Owner zugreifen können (`chmod 0600`). Nun sollte der SSH Server den entsprechenden User mit seinem Public Key „kennen“.

Wichtig: Je nach verwendeter SSH Implementation variiert das Format der Key's, d.h. sie können inkompatibel sein. PuTTY verwendet beispielsweise ein eigenes Format wenn die Key's abgespeichert werden, stellt aber andererseits ein OpenSSH kompatiblen String zur Verfügung.

Wurde der private Key dem SSH Server hinzugefügt, muss dem SSH Client noch der private Key zugewiesen werden.




**Abbildung 10: Einer PuTTY Session wird ein Private Key zugewiesen**

Wird nun die Verbindung nun Aufgebaut, schickt der Agent den key und dessen passphrase an den SSH Server – es muss kein Passwort mehr eingegeben werden:

```
Using username "administrator".
Authenticating with public key "rsa-key-20071005"
Passphrase for key "rsa-key-20071005":
Last login: Sat Oct 6 00:09:44 2007 from localhost
Fanfare!!!
You are successfully logged in to this server!!!
```

Jedoch muss noch der Passphrase des private key's eingegeben werden. Stimmt dieser Passphrase kann der private key verwendet (=entschlüsselt) werden; die Authentifizierung ist abgeschlossen.

Damit nicht bei jeder neuen SSH Verbindung der Passphrase eingegeben werden muss, gibt es den SSH-Agent. Dem SSH-Agent wird nach dessen Start ein oder mehrere private keys hinzugefügt, inklusive dem dazugehörigen Passphrase.

Unter Windows verwende ich Pageant  als SSH Agent. Nach dessen Start erscheint ein Icon im Systemtray. Dort kann mittels eines rechten Mausklicks der oder die private keys hinzugefügt werden. Wurde dies gemacht und der Passphrase korrekt eingegeben muss man bei einer erneuten SSH Verbindung **KEINEN** Passphrase (und auch kein Passwort) mehr eingeben:

```
Using username "administrator".
Authenticating with public key "rsa-key-20071005" from agent
Last login: Sat Oct 6 00:12:06 2007 from localhost
Fanfare!!!
You are successfully logged in to this server!!!

Administrator@abcitchy ~
$
```

## Die Public Key Authentifizierung Linux-to-Linux:

Unter Linux stellt OpenSSH die zu verwendenden Tools zur Verfügung.

SSH Keypair erstellen:

```
Administrator@abcitthy ~/.ssh
$ ssh-keygen.exe
Generating public/private rsa key pair.
Enter file in which to save the key (/home/Administrator/.ssh/id_rsa):
...
```

SSH Agent starten:

```
Administrator@abcitthy ~/.ssh
$ ssh-agent.exe
SSH_AUTH_SOCK=/tmp/ssh-xNNKysmfCq/agent.15804; export SSH_AUTH_SOCK;
SSH_AGENT_PID=2916; export SSH_AGENT_PID;
echo Agent pid 2916;

Administrator@abcitthy ~/.ssh
$ ssh-add.exe
Could not open a connection to your authentication agent.
```

Der Agent muss via Umgebungsvariable bekannt gemacht werden (siehe Output oben), sonst weiss der SSH Client nicht wo die Key's hinzugefügt werden können.

SSH Umgebungsvariablen setzen und Agent starten

```
Administrator@abcitthy ~/.ssh
$ SSH_AUTH_SOCK=/tmp/ssh-xNNKysmfCq/agent.15804; export SSH_AUTH_SOCK;
$ SSH_AGENT_PID=2916; export SSH_AGENT_PID;

$ ssh-add.exe
Enter passphrase for /home/Administrator/.ssh/id_rsa:
Identity added: /home/Administrator/.ssh/id_rsa (/home/Administrator/.ssh/id_rsa)
```

Verwendet man OpenSSH<sup>13</sup> gibt es ein Bash Script namens „ssh-copy-id“, welches das public key File auf den Zielrechner kopiert:

```
Administrator@abcitthy /bin
$ ssh-copy-id -i /home/Administrator/.ssh/id_rsa.pub 192.168.111.22
35
Password:
Now try logging into the machine, with "ssh '192.168.111.22'", and check in:

.ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
```

SSH Verbindung ohne Passwort:

```
Administrator@abcitthy ~/.ssh
$ ssh 192.168.111.22
Linux mideb 2.6.8-2-686 #1 Tue Aug 16 13:22:48 UTC 2005 i686 GNU/Linux

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Mon Oct 8 12:23:21 2007 from 192.168.111.24
Administrator@mideb:~$
```

Die Verbindung wurde erfolgreich hergestellt.

<sup>13</sup> Unter Cygwin war dieses Shell-Script nicht vorhanden. Ich habe das Script von einer Linux Machine auf meine Cygwin Machine kopiert.

## Agent forwarding

Möchte von einem SSH Server aus eine weitere Verbindung auf einen anderen SSH Server herstellen, ohne ein Passwort einzugeben, kann man den SSH Agent dazu verwenden, den Key weiterzuleiten. Das hat den Vorteil, dass weiterhin auf meinem Client der private key liegt und auf den SSH Server nur der public key. Natürlich muss auf jedem Server der public key des Users vorhanden sein.

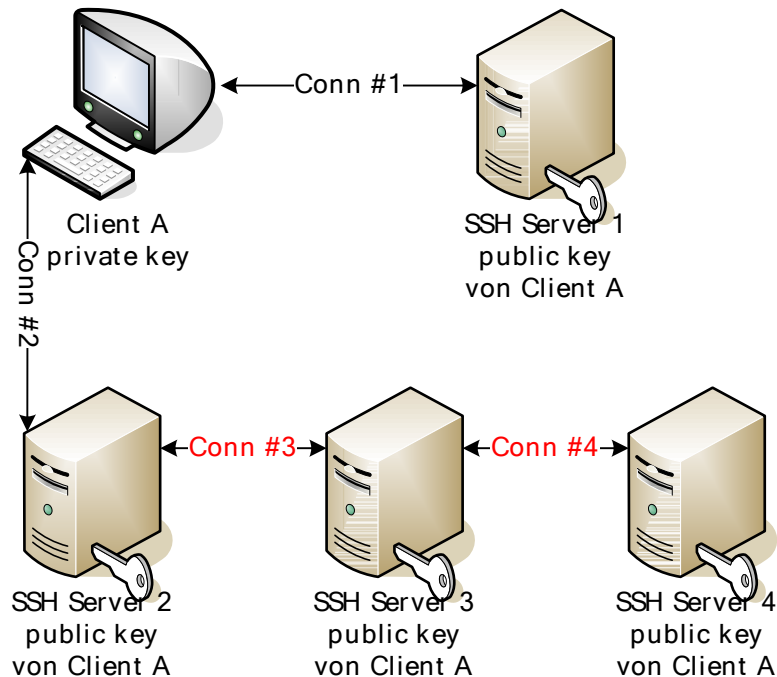


Abbildung 11: SSH Agent Verwendung

Verbindet sich Client A auf Server 2, muss kein Passwort eingegeben werden. Verbindet man sich nun von SSH Server 2 auf SSH Server 3 muss ebenfalls kein Passwort eingegeben werden, da der Agent den privaten key automatisch weiterreicht (das Gleiche gilt für die Verbindung von SSH Server 3 auf SSH Server 4).

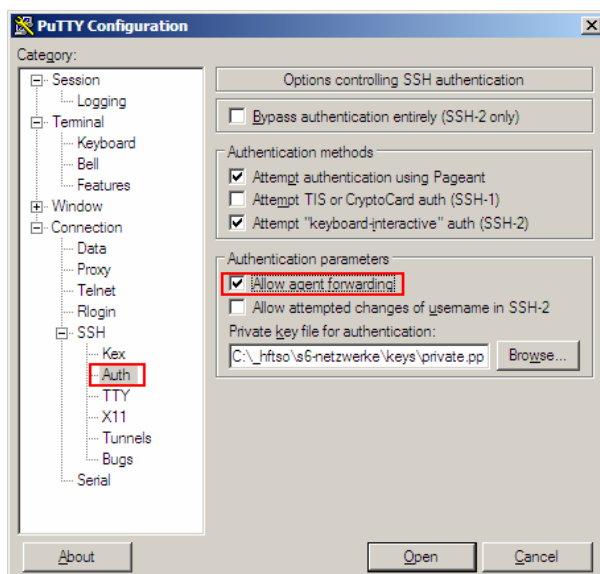


Abbildung 12: PuTTY Agent Weiterleitung

PuTTY unterstützt die Agent Weiterleitung.

**ACHTUNG:** Agent forwarding ermöglicht, dass ein User mit root Rechten auf dem Ziel SSH Server, via Unix Sockets, an den private key herankommt.

## HOSTBASED AUTHENTIFIZIERUNG

Auszug aus der sshd\_config manpage:

```
HostbasedAuthentication
    Specifies whether rhosts or /etc/hosts.equiv authentication together
    with successful public key client host authentication is allowed (host
    based authentication). This option is similar to
    RhostsRSAAuthentication and applies to protocol version 2 only. The
    default is no
```

Das SSH helper Tool ssh-keysign muss auf dem Client installiert sein<sup>14</sup>:

```
ssh-keysign is used by ssh(1) to access the local host keys
and generate the digital signature required during hostbased
authentication with SSH protocol version 2.
```

```
ssh-keysign is disabled by default and can only be enabled
in the global client configuration file /etc/ssh/ssh_config
by setting EnableSSHKeysign to 'yes'.
```

```
ssh-keysign is not intended to be invoked by the user, but
from ssh(1). See ssh(1) and sshd(8) for more information
about hostbased authentication.
```

SSH Client Konfiguration:

- Ändern des SSH Konfigurationsfiles ssh\_config (häufig unter /etc/ssh/ssh\_config):  
HostbasedAuthentication yes  
EnableSSHKeysign yes
- Setuid für das File /usr/bin/ssh-keysign (chmod u+s) setzen
- Das File /root/.shosts erstellen:

```
root@Openwrt:~# cat /root/.shosts
localhost root
127.0.0.1 root
```

SSH Server Konfiguration:

- Ändern des SSHd Konfigurationsfiles sshd\_config (häufig unter /etc/ssh/sshd\_config):  
HostbasedAuthentication yes  
IgnoreRhosts no
- Der öffentliche Machinenschlüssel des Clients (/etc/ssh/ssh\_host\_rsa\_key.pub) muss auf dem SSH Server in das File /etc/ssh/ssh\_known\_hosts2 eingefügt werden.  
Info betreffend known\_host Dateien; /etc/ssh/ssh\_known\_hosts für SSH1 RSA keys bzw. /etc/ssh/ssh\_known\_hosts2 für SSH2 RSA+DSA keys. Beispiel:

```
root@Openwrt:~# ssh-keyscan -p 443 -t rsa localhost > /etc/ssh/ssh_known_hosts2
root@Openwrt:~# cat /etc/ssh/ssh_known_hosts2
localhost,localhost.,127.0.0.1 ssh-rsa AAAAB3NzaC1yc2EAAA..<snip>..sGV56ww==
```

Client, folgende Files sind involviert:

- /root/.shosts – Username der verwendet wird bei einer Verbindung

Server, folgende Files sind involviert:

- /etc/ssh/ssh\_known\_hosts2 – hostname, key type, public key
- /etc/hosts – namensauflösung muss für alle beteiligten Hosts funktionieren

<sup>14</sup> <http://www.tiem.utk.edu/cgi-bin/man.cgi?section=8&topic=ssh-keysign>

Test mit debug informationen:

```
root@openwrt:~# ssh -vv -p 443 localhost
...
debug1: Next authentication method: hostbased
debug2: userauth_hostbased: chost localhost..
debug2: we sent a hostbased packet, wait for reply
debug1: Remote: Accepted by .shosts.
debug1: Remote: Accepted host localhost. ip 127.0.0.1 client_user root server_user root
debug1: Authentications that can continue: publickey,password,keyboard-interactive,hostbased
debug2: userauth_hostbased: chost localhost..
debug2: we sent a hostbased packet, wait for reply
debug1: Remote: Accepted by .shosts.
debug1: Remote: Accepted host localhost. ip 127.0.0.1 client_user root server_user root
debug1: Authentication succeeded (hostbased).
```

Info: Damit die Hostbased Authentifizierung funktioniert, muss die DNS Auflösung für beide Server funktionieren, sonst wird folgende Fehlermeldung ausgegeben:

```
fwpiercer:~# ssh -p 443 m00m00.no-ip.org
get_socket_address: getnameinfo 8 failed: Name or service not known
userauth_hostbased: cannot get local ipaddr/name
```

Ich habe versucht, einen SSH Server, welcher an einem privaten LAN angeschlossen ist, via Hostbased Authentifizierung auf einem entfernten SSH Server anzumelden. Der Server „kennt“ aber diesem Host aus dem privaten LAN nicht. Hier der dazugehörige Syslog Auszug:

```
Jan 12 17:00:29 openwrt auth.info /usr/sbin/sshd[691]: Connection from 80.219.162.37 port 24005
Jan 12 17:00:29 openwrt auth.debug /usr/sbin/sshd[691]: debug1: Client protocol version 2.0; client software version OpenSSH_4.3p2 Debian-9
Jan 12 17:00:29 openwrt auth.debug /usr/sbin/sshd[691]: debug1: match: OpenSSH_4.3p2 Debian-9 pat OpenSSH*
Jan 12 17:00:29 openwrt auth.debug /usr/sbin/sshd[691]: debug1: Enabling compatibility mode for protocol 2.0
Jan 12 17:00:29 openwrt auth.debug /usr/sbin/sshd[691]: debug1: Local version string SSH-2.0-OpenSSH_4.7
Jan 12 17:00:29 openwrt auth.info /usr/sbin/sshd[691]: WARNING: /etc/ssh/moduli does not exist, using fixed modulus
Jan 12 17:00:30 openwrt auth.info /usr/sbin/sshd[691]: Address 80.219.162.37 maps to fwpiercer, but this does not map back to the address - POSSIBLE BREAK-IN ATTEMPT!
Jan 12 17:00:31 openwrt auth.info /usr/sbin/sshd[691]: Failed none for root from 80.219.162.37 port 24005 ssh2
```

## LOKALER TUNNEL

Ein lokaler Tunnel ist eine Port Weiterleitung vom SSH Server zum SSH Client, d.h. der entfernte TCP Port (z.B. Port 80) wird lokal gemappt (z.B. als Port 8080).

## RDP over SSH

Ein erstes, einfaches Beispiel wie ein lokaler Tunnel funktioniert. Wir authentifizieren uns auf einem SSH Server und verbinden uns anschliessend auf einen Windows Server via RDP:

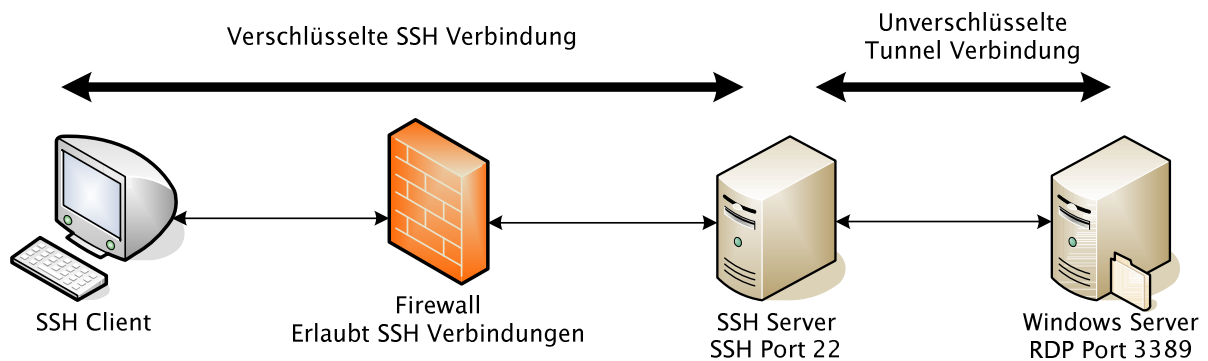


Abbildung 13: Netzwerk Übersicht

Das einzige, was wir benötigen, um den Windows Server via RDP fernzusteuern, ist eine SSH Verbindung mit einem lokalen Tunnel. Dazu benötigen wir den SSH Client PuTTY:

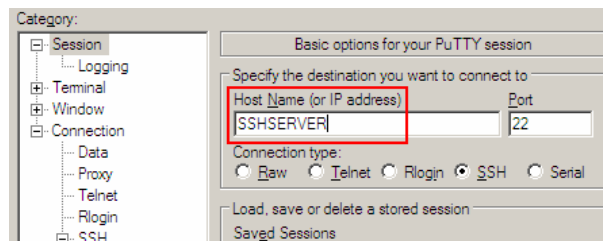


Abbildung 14: PuTTY, Verbindung herstellen

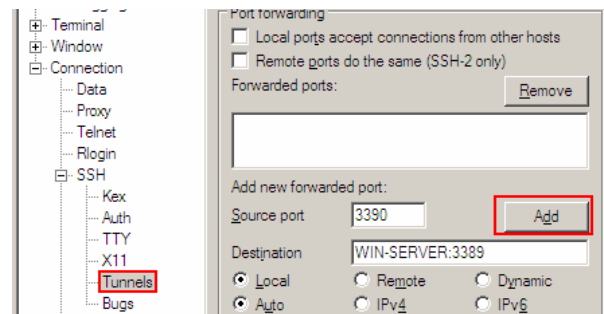


Abbildung 15: PuTTY, Tunnel erstellen

Zuerst wird eine Verbindung zum SSH Server aufgebaut (Abbildung 14). Ist dies erledigt wird vom SSH Server eine Verbindung zum Windows Server aufgebaut (Abbildung 15). Nun leitet SSH die Daten des Windows Servers von Port 3389 an den SSH Client an Port 3389. Der RDP Client kann nun eine (SSH verschlüsselte) Verbindung auf den Windows Server aufbauen:

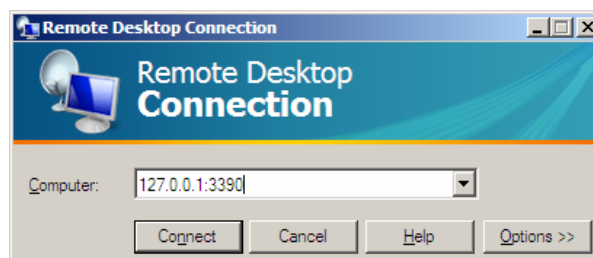


Abbildung 16: Eine RDP-over-SSH Verbindung wird aufgebaut

Info: Der Tunnel wird vom SSH Server aus erstellt und ist *nicht verschlüsselt*.

## SMB over SSH (net use)

Man kann auch eine SMB Netzwerkverbindung („net use“) über SSH herstellen. Dazu muss eine neue Loopback NIC (Netzwerkkarte) installiert werden:

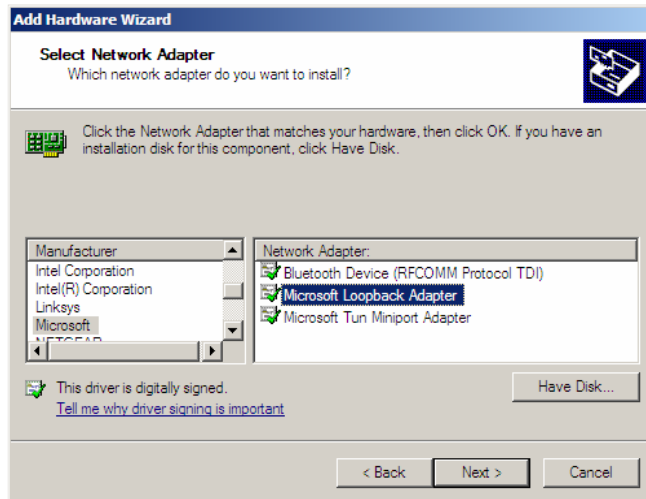


Abbildung 17: Loopback NIC installieren

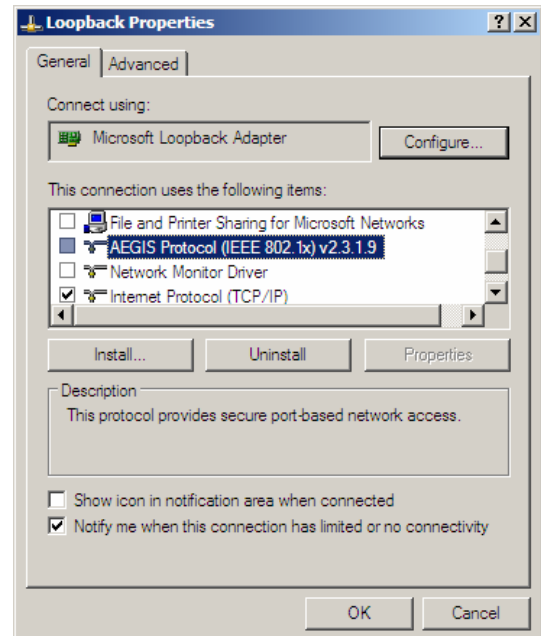


Abbildung 18: Konfiguration der Loopback NIC

- IP Adresse 10.0.0.1/24
- Nur die TCP/IP Komponente auswählen (kein "Client for Microsoft Networks", "File and Printer Sharing..." usw)
- Unter Advanced TCP/IP Settings, WINS Tab, die Option "NetBIOS over TCP/IP" deaktivieren und lmhosts lookup aktivieren. Unser LAN Adapter verwendet bereits "NetBIOS over TCP/IP" (Port 139).

Nun muss das lmhosts File (%windir%\system32\drivers\etc\lmhosts) um den remote SSH Server erweitert werden, Beispiel:

```
10.0.0.1 MEIN_SSH_SERVER
```

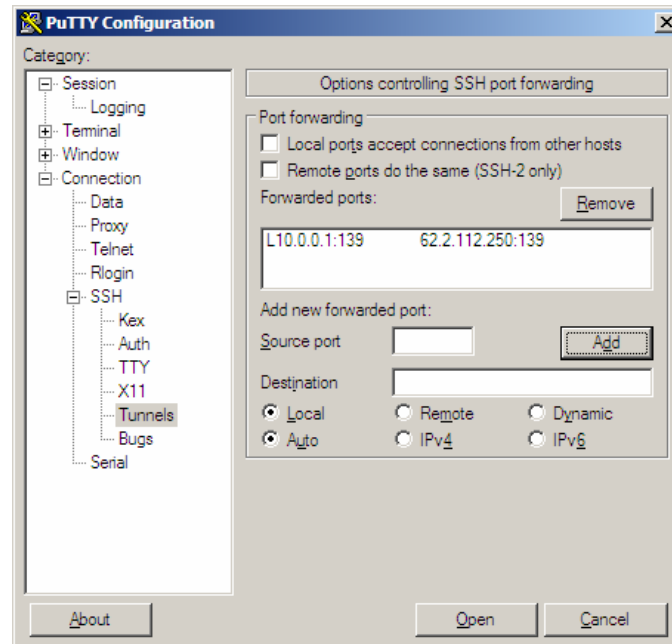
Ein "Ping MEIN\_SSH\_SERVER" sollte erfolgreich funktionieren:

```
c:\WINDOWS\system32\drivers\etc>ping MEIN_SSH_SERVER
Pinging MEIN_SSH_SERVER [10.0.0.1] with 32 bytes of data:

Reply from 10.0.0.1: bytes=32 time<1ms TTL=64
Reply from 10.0.0.1: bytes=32 time<1ms TTL=64
Reply from 10.0.0.1: bytes=32 time<1ms TTL=64
Reply from 10.0.0.1: bytes=32 time<1ms TTL=64

Ping statistics for 10.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Das Netzwerk ist nun vorbereitet und PuTTY kann gestartet werden. Es muss ein neuer lokaler Tunnel für Port 139 erstellt werden (Info: in das „Source port“ Feld kann auch ein Hostname und Port eingetragen werden):



**Abbildung 19: Lokaler SSH Tunnel**

Ist die SSH Verbindung hergestellt, kann das Netzwerklaufwerk verbunden werden:

```
C:\> net use * \\10.0.0.1\c$ /user:10.0.0.1\administrator password
```

## DYNAMIC TUNNEL

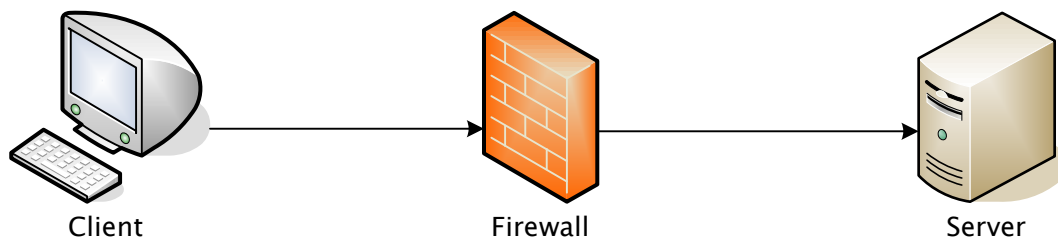
OpenSSH erlaubt die Nutzung eines eingebauten SOCKS Proxy. Zuerst ein kurzer Überblick des SOCKS Protokolls<sup>15</sup>:

*Das SOCKS Protokoll ist ein Internet-Proxy-Protokoll, das Client-Server-Anwendungen erlaubt, transparent die Dienste einer Firewall zu nutzen. SOCKS ist eine Abkürzung für "SOCKeT S".*

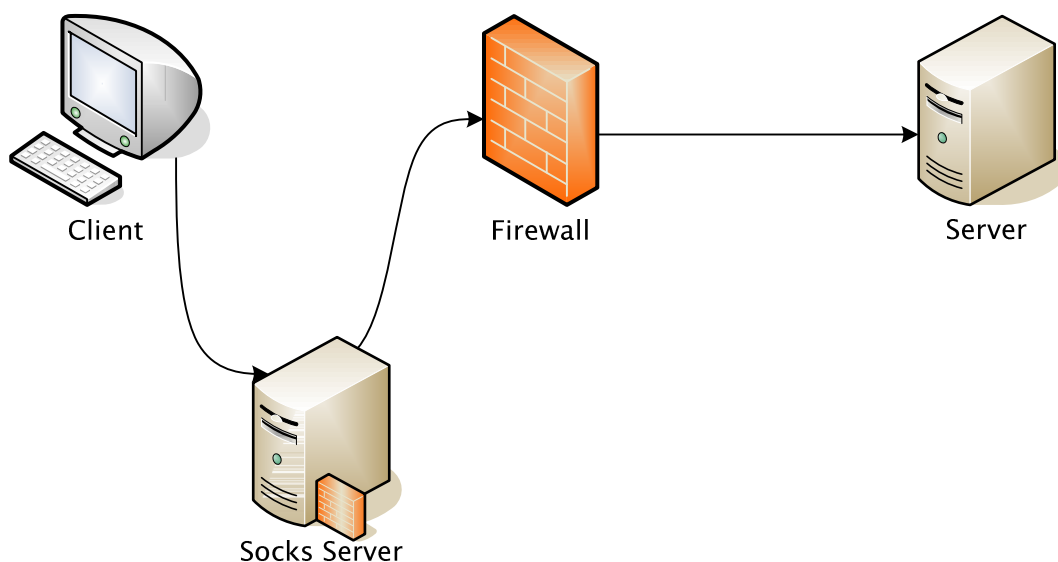
*Clients hinter einer Firewall, die eine Verbindung zu einem externen Server aufbauen wollen, verbinden stattdessen zu einem SOCKS-Proxy. Dieser Proxy-Server überprüft die Berechtigung des Clients, den externen Server zu kontaktieren und leitet die Anfrage an den Server weiter.*

*Das SOCKS-Protokoll wurde ursprünglich von NEC entwickelt (SOCKS 4). Die aktuelle Version 5 des Protokolls, wie beschrieben in RFC 1928, erweitert die vorherigen Versionen um Unterstützung für UDP, Authentifizierung, Namensauflösung am SOCKS Server und IPv6.*

*Im OSI-Schichtmodell ist es eine Zwischenschicht zwischen der Anwendungsschicht und der Transportschicht.*



**Abbildung 20: Ein Client verbindet sich zu einem Internet Server**



**Abbildung 21: Ein Server verbindet sich via SOCKS Proxy zu einem Internet Server**

Es gibt noch eine weitere SOCKS Version, Version 4a. Diese inoffizielle Version erweitert die Version 4 des SOCKS Protokolls um die Möglichkeit, einen Domain Namen (DNS) via SOCKS 4a Server auflösen zu lassen.

Laut der MAN page von SSH wird das SOCKSv4 und SOCKSv5 Protokoll unterstützt.

<sup>15</sup> <http://de.wikipedia.org/wiki/SOCKS>:

Vorteile wenn SOCKS verwendet wird:

- User Authentifizierung auf dem SOCKS Server (v5)
- Outgoing traffic kann reguliert werden
- Zentraler Punkt um Verbindungen zu überwachen

In PuTTY wird der SOCK Proxy als dynamischer Tunnel definiert. Ein Beispiel um einen SOCKS Proxy auf Port 1100 zu erstellen:

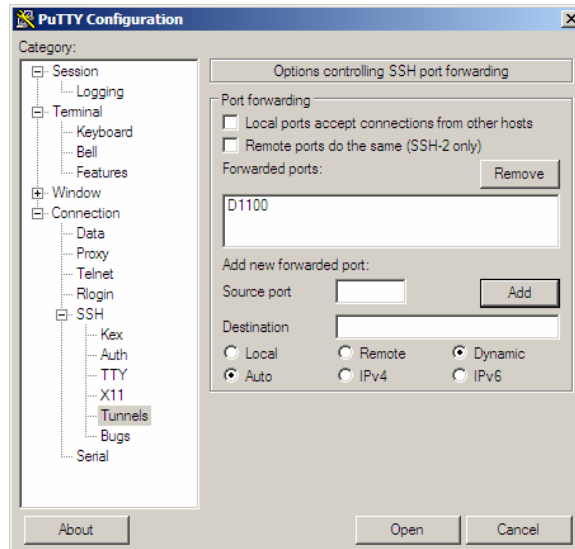


Abbildung 22: PuTTY als SOCKS Proxy



Abbildung 23: SOCKSv1

Auch der OpenSSH Client kann mit der Option „-D“ einen SOCKS Proxy verwenden.

Es gibt relativ viele Applikationen welche SOCKS Proxy Server „out of the box“ unterstützen, z.B. Internet Explorer, Mozilla Firefox, ICQ, Pidgin (Multi-Messenger Client), Azureus (Torrent Client), MiRC (IRC Client)...

## DNS Leak

Tor (ein Projekt um sich anonym im Internet zu „bewegen“) hat die DNS leak Problematik auf ihrer FAQ Seite aufgelistet<sup>16</sup>:

*The warning is:*

*Your application (using socks5 on port %d) is giving Tor only an IP address. Applications that do DNS resolves themselves may leak information. Consider using Socks4A (e.g. via privoxy or socat) instead.*

*If you are running Tor to get anonymity, and you are worried about an attacker who is even slightly clever, then yes, you should worry. Here's why.*

*The Problem. When your applications connect to servers on the Internet, they need to resolve hostnames that you can read (like tor.eff.org) into IP addresses that the Internet can use (like 209.237.230.66). To do this, your application sends a request to a DNS server, telling it the hostname it wants to resolve. The DNS server replies by telling your application the IP address.*

*Clearly, this is a bad idea if you plan to connect to the remote host anonymously: when your application sends the request to the DNS server, the DNS server (and anybody else who might be watching) can see what hostname you are asking for. Even if your application then uses Tor to connect to the IP anonymously, it will be pretty obvious that the user making the anonymous connection is probably the same person who made the DNS request.*

<sup>16</sup> <http://wiki.noreply.org/noreply/TheOnionRouter/TorFAQ#SOCKSAndDNS>

*Where SOCKS comes in. Your application uses the SOCKS protocol to connect to your local Tor client. There are 3 versions of SOCKS you are likely to run into: SOCKS 4 (which only uses IP addresses), SOCKS 5 (which usually uses IP addresses in practice), and SOCKS 4a (which uses hostnames).*

*When your application uses SOCKS 4 or SOCKS 5 to give Tor an IP address, Tor guesses that it 'probably' got the IP address non-anonymously from a DNS server. That's why it gives you a warning message: you probably aren't as anonymous as you think.*

*So what can I do? We describe a few solutions below.*

- \* If your application speaks SOCKS 4a, use it.*
- \* For HTTP (web browsing), either configure your browser to perform remote DNS lookups (see the Torify HOWTO how to do this for some versions of Firefox) or use a socks4a-capable HTTP proxy, such as Privoxy. See the Tor documentation for more information. For instant messaging or IRC, use Gaim or XChat. For other programs, consider using freecap (on Win32) or dsocks (on BSD).*
- \* If you only need one or two hosts, or you are good at programming, you may be able to get a socks-based port-forwarder like socat to work for you; see the Torify HOWTO for examples.*
- \* Tor ships with a program called tor-resolve that can use the Tor network to look up hostnames remotely; if you resolve hostnames to IPs with tor-resolve, then pass the IPs to your applications, you'll be fine. (Tor will still give the warning, but now you know what it means.)*
- \* You can use TorDNS as a local DNS server to rectify the DNS leakage.*

*See the Torify HOWTO for info on how to run particular applications anonymously.*

*If you think that you applied one of the solutions properly but still experience DNS leaks please verify there is no third-party application using DNS independently of Tor. Please see the FAQ entry on whether you're really absolutely anonymous using Tor for some examples.*

Konkret bedeutet dies, wenn wir eine Applikation mittels eines SOCKS Proxy verwenden, werden die Nutz-Daten zwar via SSH Verschlüsselt und über den SSH Server versandt, jedoch werden u.U. die DNS Abfragen NICHT getunnelt, sondern der Client löst die DNS Namen lokal auf.

Ist der DNS Server im lokalen Netz, kann ein Administrator trotzdem verfolgen, WELCHE Adressen ein User abgefragt hat (jedoch sieht der Administrator nicht WAS der User gemacht hat). SOCKS4a schafft hier Abhilfe, da diese Version auch DNS Abfrage via SOCKS Server unterstützt.

Jedoch verhalten sich nicht alle Applikationen gleich, z.B. FireFox und IE. Soll anonymes Surfen über einen SOCKS Server gewährleistet werden, sollte mit Hilfe eines Netzwerkmonitors überprüft werden, was übertragen wird.

## REMOTE TUNNEL - OUT FROM THE INSIDE

Das Gegenteil des lokalen Tunnels und der Grund, warum System Administratoren nicht erfreut sind wenn SSH verwendet wird. Ein Remote Tunnel leitet einen lokalen Port (z.B. 3389) an den SSH Server weiter. Das hat äusserst brisante Folgen: Via Internet kann auf das Lokales Netzwerk zugegriffen werden, obwohl die Firewall alle eingehenden Ports gesperrt hat – die Firewall wird ausgeschaltet.

### Remote RDP Tunnel

Wir wollen via RDP eine Workstation, welche sich in einer Firma hinter einer restriktiver Firewall befindet, (Firmen Workstation) von zu Hause aus übernehmen. Eine Übersicht der beteiligten Rechner:

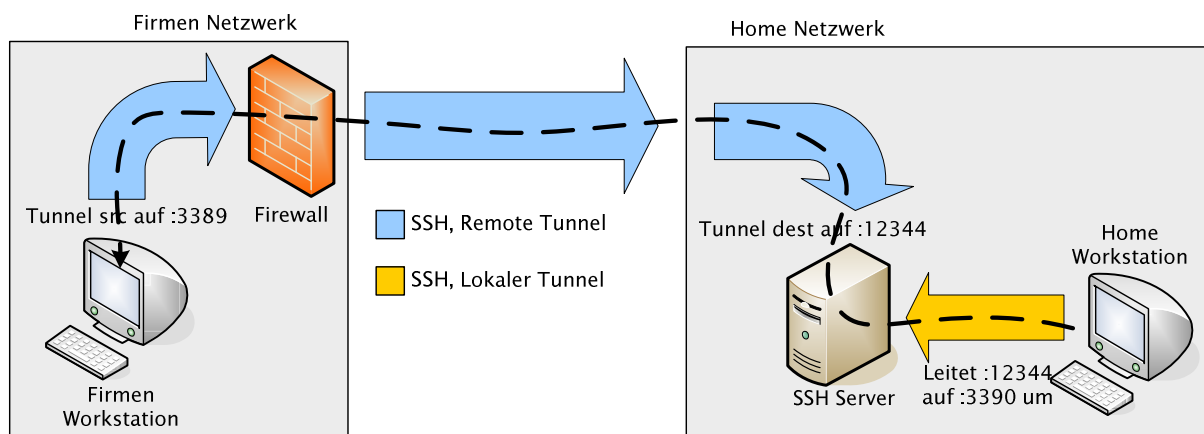


Abbildung 24: Netzwerk Übersicht Remote Tunnel

- *Firmen Workstation:* Das ist ein Windows XP Workstation in einem Geschäft. Auf diesem Rechner ist RDP aktiviert.
- *SSH Server:* Ein SSH Server.
- *Home Workstation:* Eine Workstation mit einem RDP Client. Von diesem Rechner aus wollen wir via RDP auf den Firmen Rechner zugreifen.

Die Firewall verbietet incoming Verbindungen, nur ausgehende Verbindungen sind erlaubt. Der SSH Server und die Home Workstation müssen nicht im gleichen Netz sein, das ist nur aus Gründen der Übersichtlichkeit so gelöst.

Zuerst werden die SSH Verbindungen hergestellt:

*Firmen Workstation:*

PuTTY starten und einen neuen Remote Tunnel erstellen, wir leiten den lokalen TCP Port 3389 (RDP) an den SSH Server auf Port 12344 weiter.

*Home Workstation:*

PuTTY starten und einen neuen lokalen Tunnel erstellen, wir leiten den SSH Serverport 12344 weiter auf unseren lokalen Rechner auf Port 3390.

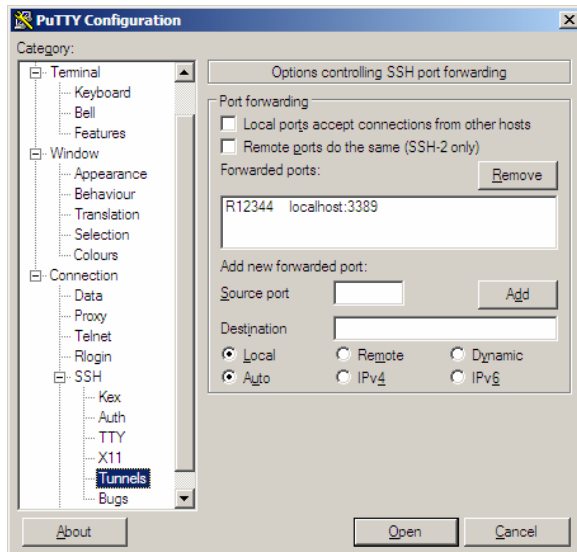


Abbildung 25: Firmen Workstation, SSH Tunnel Einstellung

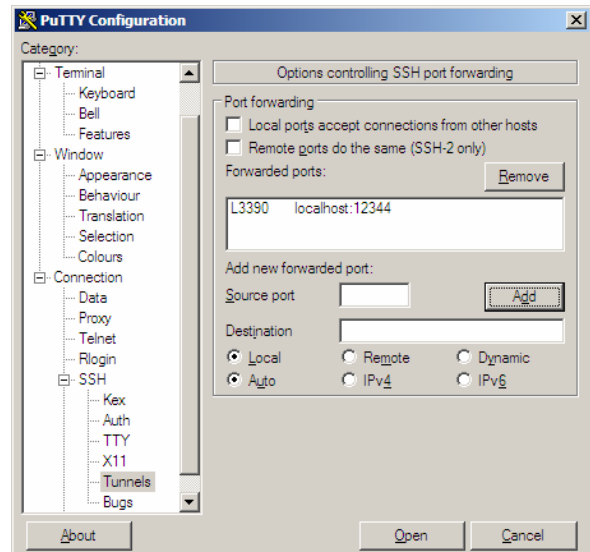


Abbildung 26: Home Workstation, SSH Tunnel Einstellung

Info: Bei einem **Remote Tunnel** bezieht sich „localhost“ auf die lokale Maschine (dort wo PuTTY ausgeführt wird), bei einem **Lokalen Tunnel** bezieht sich „localhost“ auf den SSH Server!

Nun kann auf der Home Workstation der RDP Client („mstsc.exe /console“) gestartet werden, die Destination ist nun 127.0.0.1:3390 an.

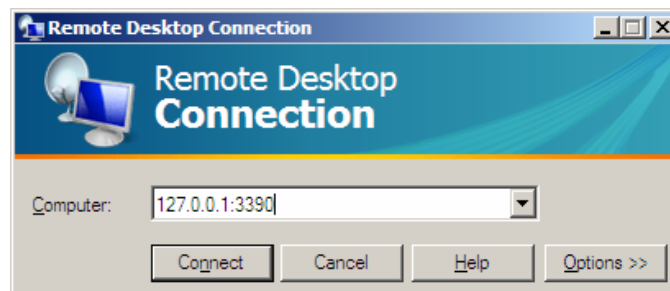


Abbildung 27: MS RDP Client

Die Firmenworkstation konnte übernommen werden, obwohl die Firewall eingehende Verbindungen abblockt!

**Fazit:** Hier wurde demonstriert, wie in ein Firmennetzwerk eingedrungen werden konnte, trotz korrekt konfigurierter Firewall. Das ganze funktioniert weil die Workstation, welche übernommen wurde, die SSH Verbindung selber aufgebaut hat und daher die Verbindung als "outgoing connection" deklariert wurde. Mit einem SSH Tunnel kann also die ganze Firewall ausser Kraft gesetzt werden.

## WINDOWS SSH SERVER MIT CYGWIN

Cygwin<sup>17</sup> ermöglicht GNU Programme unter Windows laufen zu lassen. Dazu muss das Programm mit dem Cygwin GCC Compiler neu kompiliert werden.

Via Cygwin Setup Tool kann jedoch ein Basissystem installiert werden, inkl. des SSH Daemons. Damit man den SSH Service verwenden kann, muss dieser noch konfiguriert werden.

Nach der Installation ein Cygwin Fenster öffnen (Bash Shell):

```
$ ssh-host-config
Generating /etc/ssh_config file
Privilege separation is set to yes by default since OpenSSH 3.3.
However, this requires a non-privileged account called 'sshd'.
For more info on privilege separation read /usr/share/doc/openssh/README.privsep.

Should privilege separation be used? (yes/no) yes
Warning: The following function requires administrator privileges!
Should this script create a local user 'sshd' on this machine? (yes/no) yes
Generating /etc/sshd_config file

Warning: The following functions require administrator privileges!
Do you want to install sshd as service?
(Say "no" if it's already installed as service) (yes/no) yes

which value should the environment variable CYGWIN have when
sshd starts? It's recommended to set at least "ntsec" to be
able to change user context without password.
Default is "ntsec". CYGWIN=ntsec tty

The service has been installed under LocalSystem account.
To start the service, call 'net start sshd' or 'cygrunsrv -s sshd'.

Host configuration finished. Have fun!
```

Der SSH Server muss natürlich die User und Passwörter auch kennen - wir kopieren dazu die lokalen Windows User und Gruppen in die Cygwin Umgebung:

```
mv@abcitcky /etc/skel
$ mkpasswd -cl > /etc/passwd
mv@abcitcky /etc/skel
$ mkgroup.exe --local > /etc/group
```

Info: Als Login Credentials sollte ein lokaler User verwendet werden, KEIN cached Domain User! Daher sollte auch Cygwin mit einem lokalen User gestartet werden (runas), ansonsten ist der User dem System unbekannt.

Als letzten Schritt muss noch eine Umgebungsvariable erstellt werden:

```
Set CYGWIN=ntsec tty
```

Weiter empfiehlt es sich, die Path Umgebungsvariable mit dem Cygwin bin Verzeichnis zu erweitern. Nach einem Reboot ist der SSH Server (Servicename: „CYGWIN sshd“) einsatzbereit.

<sup>17</sup> <http://www.cygwin.com/>

## BYPASS PROXY-SERVER

Ein Proxy Server dient dazu, ausgehenden Netzverkehr zentral zu überwachen. Häufig wird eine Benutzer-Authentifizierung verlangt, damit nachvollzogen werden kann, wer und wann eine Verbindung aufgebaut wurde.

Will man einen Proxy Server „umgehen“, muss der Proxyserver zuerst identifiziert werden. Der einfachste Weg dazu ist es, in den Browsereinstellungen nachzuschauen. Ist dieser Konfigurations-Bereich gesperrt oder wird ein Proxy-PAC verwendet (automatische Proxy Konfiguration) können mittels dem „netstat.exe“ Befehl die momentanen Verbindungen aufgelistet werden. Häufig laufen Proxy Server auf einem der folgenden Ports: 80, 3128, 8000, 8080.

## Proxy Authentifizierungen

Soll eine Verbindung über einen Proxy Server hergestellt werden ist eine Authentifizierung häufig obligatorisch. Dabei gibt es verschiedene Methoden, welche unter anderem im rfc2617<sup>18</sup> beschrieben werden, hier ein Auszug bekannter Authentifizierungsmethoden:

- **Basic authentication**, Klartext Übermittlung (Base64 encodiert) von User und Passwort  
Beispiel:

```
Proxy-Authenticate: Basic QWxhZGRpbjpvY2FtZQ==
```

- **Digest authentication**, übermittelt den User und Passwort nicht mehr im Klartext, sondern als Hash. Gilt jedoch nicht als sichere Methode:

*Users and implementors should be aware that this protocol is not as secure as Kerberos, and not as secure as any client-side private-key scheme. Nevertheless it is better than nothing, better than what is commonly used with telnet and ftp, and better than Basic authentication.*

Beispiel:

```
Authorization: Digest username="Mufasa",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

- **NTLM authentication** (NT LAN Manager), es gibt 2 Versionen: NTLMv1 und NTLMv2. NTLM ist ein proprietäres Protokoll von Microsoft. Diese Methode verschlüsselt die übertragenen Daten ebenfalls und ist, wenn möglich, der diggest Methode vorzuziehen. Beispiel:

```
Proxy-Authenticate: NTLM
TlRMTVNTUAACAAAADAAASAgAAoChBYEW9qhKAAAAAAAAAAAAAAAAWAAAA
-Negotiate authentication
```

Ein Auszug aus MSDN<sup>19</sup>:

*If both the server and client are using Windows 2000 or later, Kerberos authentication is used. Otherwise NTLM authentication is used. Kerberos is available in Windows 2000 and later operating systems and is considered to be more secure than NTLM authentication. For Negotiate authentication to function correctly, several exchanges must take place on the same connection. Therefore, Negotiate authentication cannot be used if an intervening proxy does not support keep-alive connections.*

In der Praxis wird häufig Basic und NTLM Authentifizierung verwendet.

<sup>18</sup> <http://www.ietf.org/rfc/rfc2617.txt>

<sup>19</sup> <http://msdn2.microsoft.com/en-us/library/aa383144.aspx>

Um die Authentifizierungsmethode eines Proxy Servers herauszufinden, kann der header des Proxy Servers angeschaut werden. Ich verwende cURL<sup>20</sup> um eine HTTP Anfrage an Google via Proxy Server zu stellen und gebe dann den HTTP Header aus:

```
c:\TEMP\curl-7.17.1>curl -i -x proxy.firma.ch:3128 www.google.com
HTTP/1.0 407 Proxy Authentication Required
Mime-Version: 1.0
Date: Tue, 04 Dec 2007 09:23:17 GMT
Content-Type: text/html
Content-Length: 237
Expires: Tue, 04 Dec 2007 09:23:17 GMT
Proxy-Authenticate: NTLM
Proxy-Authenticate: Basic realm="SQUID v2.5 proxy.firma.ch"
Proxy-Connection: close

<HTML><HEAD>
<TITLE>ERROR: Cache Access Denied</TITLE>
<META http-equiv="refresh" content="0; url=http://proxy.firma.ch/cgi-bin/squid-
CacheAccessDenied.cgi?c=10.226.64.65&u=http://www.google.com&s=squid/2.5.STABLE10">
</HEAD></HTML>
```

Ein „407 Proxy Authentication Required“ Fehler wird im RFC2616<sup>21</sup> beschrieben:

#### 10.4.8 407 Proxy Authentication Required

*This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (section 14.33) containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (section 14.34). HTTP access authentication is explained in "HTTP Authentication: Basic and Digest Access Authentication" [43].*

#### 14.33 Proxy-Authenticate

*The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.*

*Proxy-Authenticate = "Proxy-Authenticate" ":" 1#challenge*

*The HTTP access authentication process is described in "http Authentication: Basic and Digest Access Authentication" [43]. Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the current connection and SHOULD NOT be passed on to downstream clients. However, an intermediate proxy might need to obtain its own credentials by requesting them from the downstream client, which in some circumstances will appear as if the proxy is forwarding the Proxy-Authenticate header field.*

Test ob eine Verbindung durch einen Proxy Server funktioniert:

```
c:\TEMP\curl-7.17.1>curl -g --head -U user:password --proxy-basic -x proxy.firma.ch:3128
http://www.google.com
HTTP/1.0 302 Moved Temporarily
Location: http://www.google.ch/
Cache-Control: private
Set-Cookie: PREF=ID=11b3b16ec0d10514:TM=1196761626:LM=1196761626:S=5GumpUeOqvH1VD6H;
expires=Thu, 03-Dec-2009 09:47:06 GMT; path=/; domain=.google.com
Content-Type: text/html
Content-Length: 218
Date: Tue, 04 Dec 2007 09:47:06 GMT
Proxy-Connection: keep-alive
```

Diese Verbindung hat nach Angabe des Users und dessen Passworts funktioniert.

<sup>20</sup> curl - Client for URLs, <http://curl.haxx.se/>

<sup>21</sup> <http://www.ietf.org/rfc/rfc2616.txt>

## Connect-NTLM Proxy

Das Tool connect-ntlm<sup>22</sup> bietet, wie der Name schon sagt, NTLM Authentifizierung (aka "Integrated Windows Authentication"), jedoch nur NTLMv1 und nicht NTLMv2.

Ist Cygwin installiert, können alle OpenSSH Tools verwendet werden, inklusive dem SSH Client. Die Konfiguration des SSH Clients wird über das File ~/.ssh/config (~ entspricht dem Home Verzeichnis) gesteuert und wurde folgendermassen abgeändert, damit es mit connect-ntlm zusammenarbeitet:

```
Host *
  ProxyCommand connect-ntlm -H NTDOMAIN/USERNAME@PROXY_SERVER:3128 %h %p
  ServerAliveInterval 10
  Port 443
```

Anzumerken ist das NTDOMAIN der NetBIOS Name der Domäne ist, nicht der FQDN! %h wird während der Laufzeit durch den Zielhost ersetzt, %p wird durch den Zielport ersetzt.

Ein Beispiel, wie der SSH Client den Proxy Command verwendet (Debugmode):

```
$ ssh www.rootshell.be -v
OpenSSH_4.7p1, OpenSSL 0.9.8g 19 oct 2007
debug1: Reading configuration data /home/USERNAME/.ssh/config
debug1: Applying options for *
debug1: Executing proxy command: exec connect-ntlm -H domain/USERNAME@proxy.firma.ch:3128
www.rootshell.be 443
debug1: permanently_drop_suid: 400
debug1: identity file /home/USERNAME/.ssh/identity type -1
debug1: identity file /home/USERNAME/.ssh/id_rsa type -1
debug1: identity file /home/USERNAME/.ssh/id_dsa type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.6
debug1: match: OpenSSH_4.6 pat OpenSSH*
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_4.7
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: kex: client->server aes128-cbc hmac-md5 none
debug1: SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192) sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host '[www.rootshell.be]:443' is known and matches the RSA host key.
debug1: Found key in /home/.ssh/known_hosts:1
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: publickey
debug1: Trying private key: /home/USERNAME/.ssh/identity
debug1: Trying private key: /home/USERNAME/.ssh/id_rsa
debug1: Trying private key: /home/USERNAME/.ssh/id_dsa
debug1: Next authentication method: keyboard-interactive
debug1: Authentications that can continue: publickey,password,keyboard-interactive
debug1: Next authentication method: password
USERNAME@www.rootshell.be's password:
```

Ein einfacher Test, ob die Authentifizierung am Proxyserver funktioniert, wird mit folgendem Befehl ersichtlich:

```
$ connect-ntlm -H domain/USERNAME @proxy.firma.ch:3128 www.rootshell.be 443
```

<sup>22</sup> <http://legacy.not404.com/cgi-bin/trac.fcgi/wiki/SSHTunnel>

Leider habe ich PuTTY (resp. plink) nicht überreden können mit connect-ntlm zusammen zuarbeiten:

```
C:\>plink -v test
Looking up host "www.rootshell.be"
Writing new session log (SSH raw data mode) to file: putty.log
Starting local proxy command: connect-ntlm -H domain/USERNAME@proxy.firma.ch:3128
www.rootshell.be 443
We claim version: SSH-2.0-PuTTY_Snapshot_2007_11_15:r7773
Server unexpectedly closed network connection
FATAL ERROR: Server unexpectedly closed network connection
```

Test ist eine gespeicherte PuTTY Session die einen SSH Server auf Port 443 erreichen will. Die Proxy-Einstellung von PuTTY (connect-ntlm -H domain/USERNAME@firma.proxy.ch:3128 %host %port):

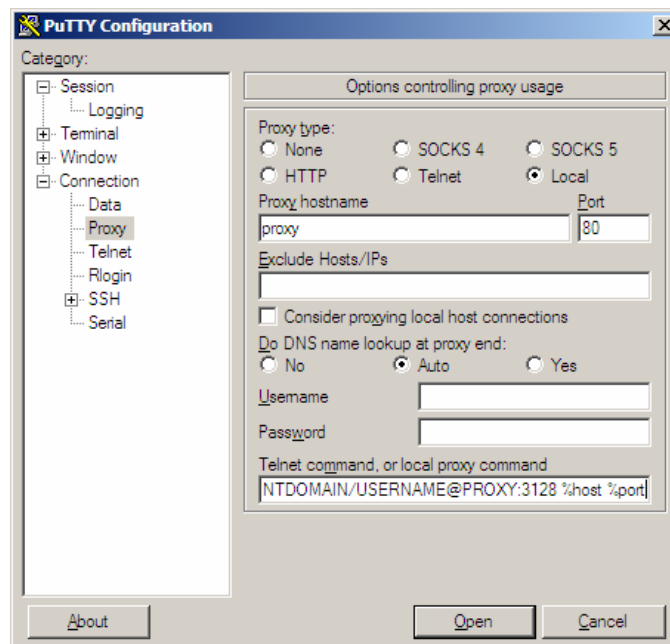


Abbildung 28: PuTTY Proxyeinstellungen

Bemerkung: Im Gegensatz zum OpenSSH Client wird hier nicht %h sondern %host als Platzhalter verwendet, das Gleiche gilt für den Port.

## NTLMMaps

Eine Alternative zu connect-ntlm ist NTLMMaps v0.9.9.0.1<sup>23</sup>. Dieses Python Script ermöglicht ebenfalls eine NTLM Authentifizierung, konfiguriert wird das Ganze mit der Datei server.cfg. Hier die wichtigsten Einstellungen:

```
[GENERAL]
LISTEN_PORT:5865
PARENT_PROXY:proxy.firma.ch
PARENT_PROXY_PORT:3128

[NTLM_AUTH]
NT_DOMAIN:domain
USER:USERNAME
PASSWORD:your_nt_password
LM_PART:1
NT_PART:0
NTLM_FLAGS: 06820000
NTLM_TO_BASIC:0
```

Starten des NTLM Services:

```
C:\> main.py
NTLM authorization Proxy Server v0.9.9.0.1
Copyright (C) 2001-2004 by Dmitry Rozmanov and others.
Now listening at wcwe00k2 on port 5865
```

Als Proxy Server wird der HTTP Proxy „localhost:5865“ verwendet.

Verbinden mit plink:

```
C:\> plink -v -load test2
Looking up host "www.rootshell.be"
Writing new session log (SSH raw data mode) to file: putty.log
Connecting to 127.0.0.1 port 5865
Server version: SSH-2.0-OpenSSH_4.6
Using SSH protocol version 2
We claim version: SSH-2.0-PuTTY_Snapshot_2007_11_15:r7773
Doing Diffie-Hellman group exchange
Doing Diffie-Hellman key exchange with hash SHA-256
Host key fingerprint is:
ssh-rsa 2048 e8:5e:e1:3b:61:7a:45:5d:bc:f6:33:c5:61:5d:0b:16
Initialised AES-256 SDCTR client->server encryption
Initialised HMAC-SHA1 client->server MAC algorithm
Initialised AES-256 SDCTR server->client encryption
Initialised HMAC-SHA1 server->client MAC algorithm
login as:
```

<sup>23</sup> <http://ntlmmaps.sourceforge.net/>

## BYPASS FIREWALLS

Eine Firewall hat die Aufgabe, das interne Netzwerk gegenüber einem externen Netzwerk (z.B. das Internet) zu schützen.

Es gibt 3 Generationen von Firewalls:

*1te Generation: Paket Filters* – Jedes Paket wird einzeln überprüft, die Firewall hat keine Ahnung, ob ein Paket Teil eines Stream ist oder nicht.

*2te Generation: Statefull Filters* – Die Firewall „weiss“, ob das momentane Paket Teil einer bestehenden Verbindung ist oder das Paket eine neue Verbindung aufgebaut. Statefull Filters können u.A. DOS Attacken verhindern.

*3te Generation: Application Layer* – Die Firewall „kennt“ Protokolle und kann daher entscheiden, ob das Datenpaket wirklich transportiert, was es vorgibt, z.B. über Port 80 kann wirklich nur HTTP Traffic passieren.

Es gibt vereinfacht gesagt zwei Arten von Firewalls: Solche, die den application layer kennen und solche die „nur“ den network layer kennen.

*Packet filtering firewall:* Diese Firewall arbeitet im Layer 3 des OSI TCP/IP Modells (Network Layer). Diese Firewall analysiert folgende 5 Informationen eines Pakets: Source IP address, Source port, Destination IP address, Destination port und IP protocol (TCP or UDP). Anhand der konfigurierten Regeln wird das Paket durchgelassen oder gesperrt.

Weitere Namen: State-full inspection firewall, network layer firewall.

*Application aware firewall:* Diese Firewall überprüft, ob die gesendeten Daten dem Protokoll entsprechen und arbeitet auf Layer 5-7 des OSI TCP/IP Modells. So ist beispielsweise auf Port 80 nur HTTP traffic erlaubt, anderer Datenverkehr nicht.

Weitere Namen: Application Firewall, Proxy Firewall, Application Gateway, Application Proxy Firewall.

Eine Packet filtering firewall kann also ohne grösseren Aufwand mit einem Tunnel „durchbort“ werden, z.B. wenn der SSH Server auf Port 80 läuft wird diese Verbindung funktionieren, eine „Application Aware“ Firewall würde dies bemerken (kein HTTP Traffic auf Port 80) und diese Verbindung sperren.

Jedoch gibt es auch die Möglichkeit ein anderes Protokoll vorzutauschen, wie das folgende Kapitel zeigt.

## HTTPTunnel

HTTPTunnel ist ein Tool bestehend aus einem Serverteil (hts) und Clientteil (htc). Der HTTPTunnel Client kapselt die Nutzdaten in ein HTTP konformes Format während der HTTPTunnel Server aus den HTTP Daten die Nutzdaten wieder extrahiert. Wird im Netzwerk eine transparente Firewall verwendet (wie an der HFTSO) eignet sich HTTPTunnel sehr gut, da hier keine Authentifizierung gegenüber dem Proxy Server benötigt wird.

Ich habe eine Windows Version von HTTPTunnel mit Cygwin erstellt.

Ein Beispiel, wie via HTTPTunnel auf einen SSH Server zugegriffen werden kann:

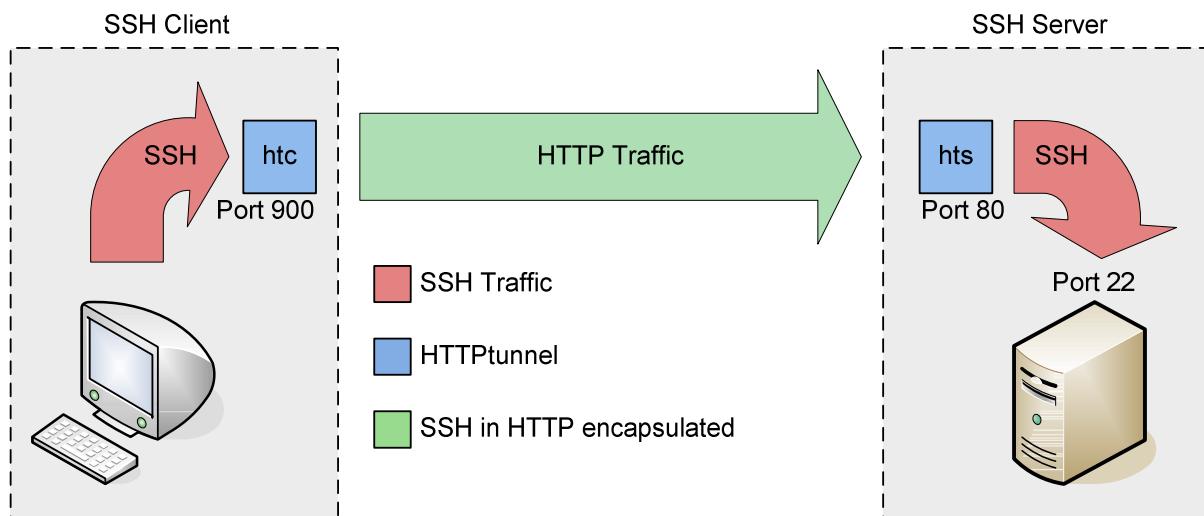


Abbildung 29: Verbindungsaufbau via HTTPTunnel

Auf dem SSH Server wird der HTTPTunnel Server (hts) gestartet. Hts erwartet Anfragen auf Port 80 und leitet die Anfragen an localhost:22 weiter, dem SSH Port. Hinweis: Normalerweise laufen Webserver auf Port 80, diese müssen natürlich gestoppt werden, wenn hts verwendet wird.

```
$ hts --forward-port localhost:22 80
```

Der HTTPTunnel Client leitet den lokalen Port 900 an den HTTPTunnel weiter

```
C:\> htc --forward-port 900 _SSH_SERVER_:80
```

Jetzt kann PuTTY gestartet werden, als Zielrechner muss localhost Port 900 angegeben werden, damit via HTTPTunnel eine Verbindung aufgebaut wird.

Gibt es Probleme mit der Verbindung, kann der Debug Modus verwendet werden:

```
$ hts --no-daemon -D4 --forward-port localhost:22 80
```

```
C:\> htc --no-daemon -D4 -F 900 _SSH_SERVER_:80
```

Die Option -Dx aktiviert den Debug Modus wobei x zwischen 1-5 sein kann und den Debuglevel angibt.

Der HTTPTunnel Client unterstützt auch Proxyserver (--proxy-authorization USER:PASSWORD), jedoch werden NTLM Proxys (z.B. MS ISA Server) nicht unterstützt sondern nur Basic Authentifikation!

```
htc -F 900 -P proxy.firma.ch:3128 --proxy-authorization _USER_:_PASSWORD_ _HTS-SERVER_:80
```

Es gibt zusätzliche Tools wie z.B. das Python Tool ntlmaps, welche als zusätzliche Proxy Server dienen können und zusätzlich noch die NTLM Authentifikation ermöglichen.

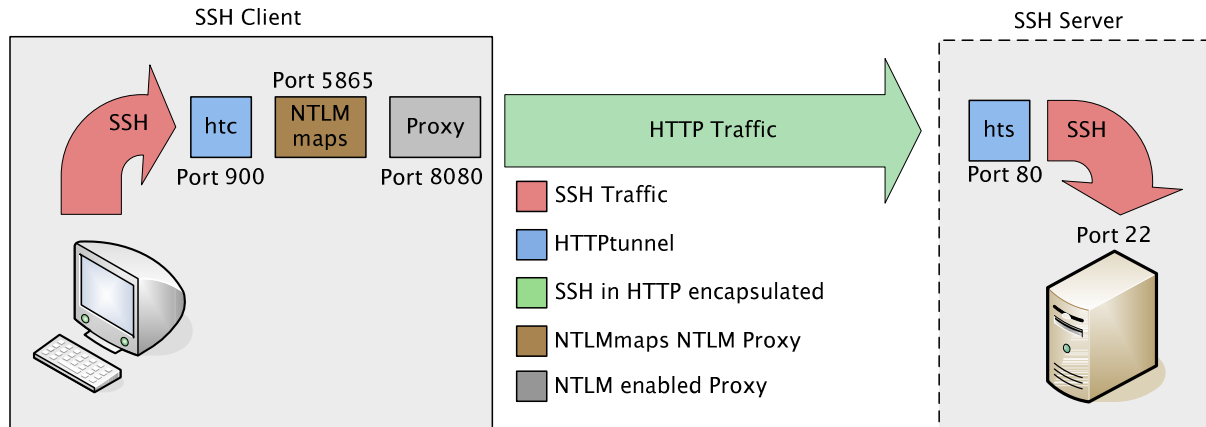


Abbildung 30: Verbindungsaufbau via HTTPtunnel und NTLM Maps

Hier ein Netzwerk dump eines HTTPtunnel Datenpakets:

```
POST /index.html?crap=1190374766 HTTP/1.1
Host: 62.2.112.250:80
Content-Length: 102400
Connection: close
...*...SSH-2.0-PuTTY_Snapshot_2007_06_05:r7610
...|.....0/..I.
0?/.7....diffie-hellman-group-exchange-sha256,diffie-hellman-group-exchange-sha1,diffie-
hellman-group14-sha1,diffie-hellman-group1-sha1,rsa2048-sha256,rsa1024-sha1...ssh-rsa,ssh-
dss...aes256-ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-
ctr,aes128-cbc,3des-ctr,3des-cbc,blowfish-ctr,blowfish-cbc,arcfour256,arcfour128...aes256-
ctr,aes256-cbc,rijndael-cbc@lysator.liu.se,aes192-ctr,aes192-cbc,aes128-ctr,aes128-cbc,3des-
ctr,3des-cbc,blowfish-ctr,blowfish-cbc,arcfour256,arcfour128...hmac-sha1,hmac-sha1-96,hmac-
md5...zlib,none...zlib,none.....q..nw.....H....]..I].
..A/"..K....n.....J.k.....4<awT...[b.#.GIFS.V.)1...(t.xml...p.P..c.%5j.e....#.dn{.g..
V.8.{...*.....
...sh..d./{F.....fI8..JEmo.....F...JgCV..c.o.V|.q...I.N:...?.....H.....u...'....r.!+...L
?\\...LBap..wJ.b.W.4-.....}....5...22C....D....
<....>
```

## DNS Tunnel (IP-over-DNS)

Der grosse Vorteil eines DNS Tunnels ist, dass diese Tunneling Variante an vielen Orten verwendet werden kann. Dem stehen aber auch zwei grosse Nachteile gegenüber:

- Relativ aufwändig aufzusetzen (DNS Eintrag)
- Schlechte Performance, ca. 5-15 k/s

Eine aktive IP-over-DNS Applikation ist Iodine<sup>24</sup>, ein Auszug aus dem Readme:

HOW TO USE:

Server side:

To use this tunnel, you need control over a real domain (like mytunnel.com), and a server with a public IP number (not behind NAT) that does not yet run a DNS server. Then, delegate a subdomain (say, tunnel1.mytunnel.com) to the server. If you use BIND for the domain, add these lines to the zone file:

```
tunnel1host    IN      A       10.15.213.99
tunnel1        IN      NS      tunnel1host.mytunnel.com.
```

Do not use CNAME instead of A above.

If your server has a dynamic IP, use a dynamic dns provider:

```
tunnel1        IN      NS      tunnel1host.mydyndnsprovider.com
```

Ich verwende No IP (<http://no-ip.org>) als DNS Provider (DynDNS). Dieser Provider unterstützt keine manuellen NS Einträge – genau wie andere DynDNS Dienst wie z.B. [dyndns.org](http://dyndns.org). Daher habe ich leider keine Möglichkeit Iodine zu testen!

---

<sup>24</sup> <http://code.kryo.se/iodine/>

## ICMP Tunnel

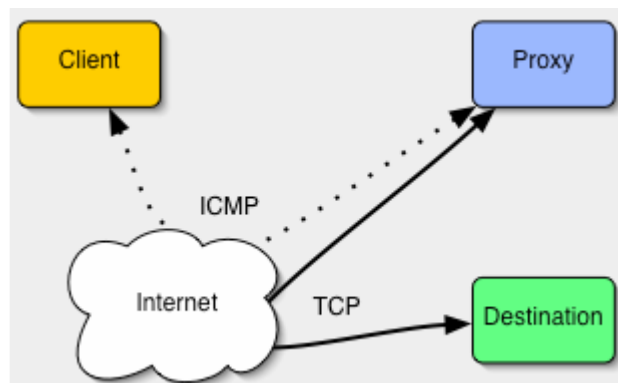
Ein ICMP Tunnel versteckt Nutzdaten in einem ICMP Paket (Ping request und replies). Ich verwende ptunnel<sup>25</sup> welches mit Hilfe von Cygwin auf Windows portiert wurde<sup>26</sup>. Die Beschreibung von der ptunnel Webseite:

*Ptunnel is not a feature-rich tool by any means, but it does what it advertises. So here is what it can do:*

- \* Tunnel TCP using ICMP echo request and reply packets*
- \* Connections are reliable (lost packets are resent as necessary)*
- \* Handles multiple connections*
- \* Acceptable bandwidth (150 kb/s downstream and about 50 kb/s upstream are the currently measured maximas for one tunnel, but with tweaking this can be improved further)*
- \* Authentication, to prevent just anyone from using your proxy*

*So what do you need for all this to work?*

- \* One computer accessible on the internet that is not firewalled (or at least allows incoming ICMP packets)*
- \* A computer to act as the client (this will usually be your laptop, on the go..)*
- \* Root access, preferably on both computers*
- \* A posix-compliant OS, with libpcap (for packet capturing)*



**Abbildung 31: Funktionsweise von ptunnel**

Linux Server, starten des daemons (br-lan ist das Netzwerk device):

```
root@Openwrt:/# ptunnel -c br-lan > /dev/null&
```

Windows Client Verbindung herstellen:

```
c:\>ptunnel.exe -lp 8000 -p m00m00.no-ip.org -da 127.0.0.1 -dp 443
[inf]: Starting ptunnel v 0.62.
[inf]: (c) 2004-2005 Daniel Stuedle, daniels@cs.uit.no
        windows Port by michu / www.neophob.com
[inf]: HINT: start ptunnel with "-h" parameter to view help and windows winPcap devices
[inf]: Relaying packets from incoming TCP streams.
[inf]: Incoming connection.
[evt]: No running proxy thread - starting it.
[inf]: Ping proxy is listening in privileged mode.
```

Nun kann via localhost:8000 eine SSH Verbindung via ICMP aufgebaut werden.

<sup>25</sup> <http://www.cs.uit.no/~daniels/PingTunnel/>

<sup>26</sup> <http://www.neophob.com/serendipity/index.php?/archives/128-PingTunnel-for-Windows-ICMP-tunnel.html>

Zwingend dazu ist, dass der ICMP traffic nicht gefiltert wird, ein einfacher Ping Befehl zeigt dies auf:

```
c:\>ping m00m00.no-ip.org
Pinging m00m00.no-ip.org [80.219.162.37] with 32 bytes of data:

Reply from 80.219.162.37: bytes=32 time=4ms TTL=64
Reply from 80.219.162.37: bytes=32 time=3ms TTL=64
```

Beispiel eines erfolgreichen Ping Versuches -> ICMP wird nicht gefiltert.

```
c:\>ping m00m00.no-ip.org
Pinging m00m00.no-ip.org [80.219.162.37] with 32 bytes of data:

Request timed out.
Request timed out.
```

Beispiel eines erfolglosen Ping Versuches -> ICMP wird gefiltert.

Wichtig zu erwähnen ist noch, dass der Windows port die Library WinPcap<sup>27</sup> benötigt:

*WinPcap is the industry-standard tool for link-layer network access in Windows environments: it allows applications to capture and transmit network packets bypassing the protocol stack, and has additional useful features, including kernel-level packet filtering, a network statistics engine and support for remote packet capture.*

*WinPcap consists of a driver, that extends the operating system to provide low-level network access, and a library that is used to easily access the low-level network layers. This library also contains the Windows version of the well known libpcap Unix API.*

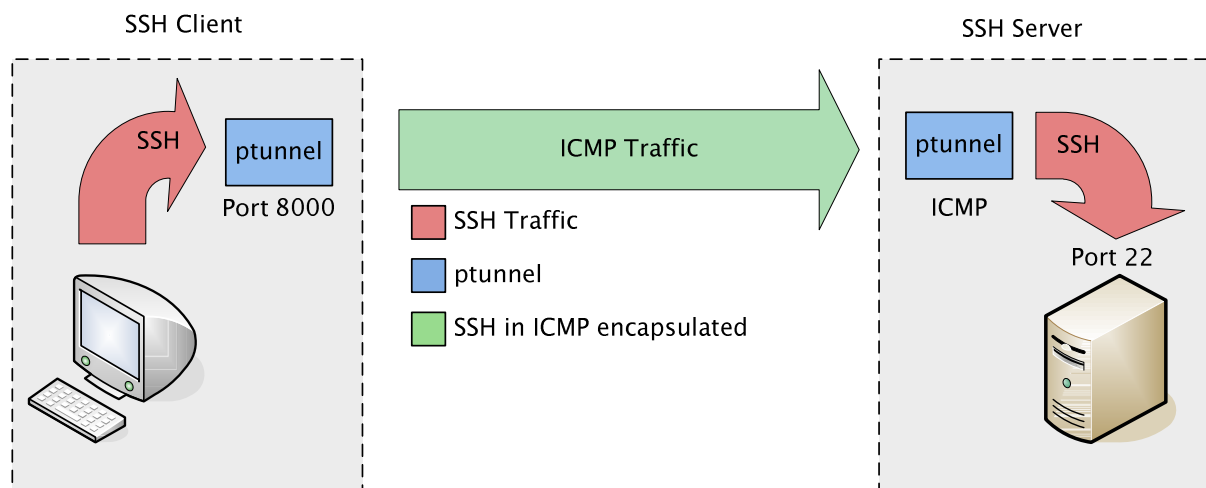


Abbildung 32: Verbindungsaufbau via ptunnel

Hinweis:

Wird auf dem Server iptables verwendet, muss eine neue Regel erstellt werden, welche erlaubt ICMP zu versenden:

```
root@openwrt:/# iptables -I OUTPUT 1 -p icmp -j ACCEPT
```

<sup>27</sup> <http://www.winpcap.org/>

## UNDER ATTACK – (ANTI) SSH TOOLS

Wenn die Verschlüsselung von SSH so sicher ist, wie ist es dann angreifbar? Der Angreifer muss sich also Zugang zu einem der Systeme (Client oder Server) verschaffen. Oder es bleibt noch die „Brute Force“-Methode übrig, welche aber Zeit und Glück in Anspruch nimmt. Die „Brute Force“-Methoden basieren nicht auf einem Implementationsfehler von SSH sondern sind ist ein bekanntes Problem von Passwort-Authentifizierung.



Hier eine Auswahl von Tools, welche das SSH Protokoll (oder SSH Produkte) angreifen oder dieses beschützen:

- DSniff v2.4<sup>28</sup> Package, Sshmitm (ehemals SShow) - monkey-in-the-middle attacks against redirected SSHmonkey-in-the-middle attacks against redirected SSH sessions.
- Guesswho v0.44<sup>29</sup> - SSH brute force Tool
- SSHatter v0.6<sup>30</sup> - SSH brute force Tool. Ein Perl Script, welches ein Passwortliste benötigt. Multi-thread fähig.
- THC Hydra v5.4<sup>31</sup> - A very fast network logon cracker which support many different services
- ScanSSH v2.0<sup>32</sup> – Sucht nach SSH Server
- SSH-Privkey-Crack v0.3<sup>33</sup> - private key brute forcer
- PuTTY Private Key Crack v0.5<sup>34</sup> - PuTTY private key brute forcer
- Sshutout v1.0.5<sup>35</sup> - ssh brute force attack blocker

Ich habe einen Testuser namens „testuser“ mit dem Passwort „password“ erstellt. Dieses Passwort ist auch in der Passwort-Liste vorhanden welche die Brute Force Tools verwenden.

### DSniff v2.4

Ermöglicht relativ einfach eine MITM Attacke zu starten, jedoch wird nur SSHv1 unterstützt, daher habe ich das Tool nicht ausprobiert.

**FAZIT:** Nur SSH Version 1 unterstützt, daher kaum wirksam resp. praktischer Nutzen!

<sup>28</sup> <http://monkey.org/~dugsong/dsniff/>

<sup>29</sup> <http://www.vulnerabilityassessment.co.uk/guesswho.htm>

<sup>30</sup> <http://www.nth-dimension.org.uk/downloads.php?id=34>

<sup>31</sup> <http://freeworld.thc.org/thc-hydra/>

<sup>32</sup> <http://monkey.org/~provos/scanssh/>

<sup>33</sup> <http://www.neophob.com/serendipity/index.php?/archives/123-SSH-Private-Key-cracker.html>

<sup>34</sup> <http://www.neophob.com/serendipity/index.php?/archives/127-PuTTY-Private-Key-cracker.html>

<sup>35</sup> <http://www.techfinesse.com/sshutout/sshutout.html>

## GuessWho v0.44:

Dieses Linux Tool muss selbst kompiliert werden und erzeugt ein binäres File namens „b“.

```
Usage: ./b <-l login> <-h host> [-p port] <-1|-2> [-N nthreads] [-n ntries]
Use -1 for producer/consumer thread model, -2 for dumb parallelism. Passwds go on stdin. :)
```

Testlauf:

```
mideb:/home/ssh/guess-who# ./b -l Administrator -h 192.168.111.22 -2 < ./passwords
[ 00268 ][ 00415 ][ 00000000.645782 ][ Administrator ][          hansolo ]
```

Da passierte einfach nichts... Deshalb habe ich versucht via SSH Client auf den gerade angegriffenen SSH Server zu Verbinden:

```
mideb:~# ssh -v testuser@localhost
OpenSSH_3.8.1p1 Debian-8.sarge.6, OpenSSL 0.9.7k 05 Sep 2006
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Connecting to localhost [127.0.0.1] port 22.
debug1: Connection established.
debug1: identity file /root/.ssh/identity type -1
debug1: identity file /root/.ssh/id_rsa type -1
debug1: identity file /root/.ssh/id_dsa type -1
ssh_exchange_identification: Connection closed by remote host
```

Diese Fehlermeldung sagt aus, dass zu viele Connections auf dem SSH Server offen sind. Ich habe die zwei Multi-threading Optionen ausprobiert, beide ohne Erfolg.

**FAZIT:** Ich habe keine SSH Verbindung zustande gebracht, das Programm schien einfach zu warten oder der SSH Server blockte die Verbindung aufgrund zu vieler versuchten Verbindungen. Ich habe auch noch einen Debug Build erstellt, aber auch dies hat mich nicht weitergebracht.

## SSHatter v0.6

Nach zahlreichen Nachinstallationen von Perl Modulen war SSHatter einsatzbereit:

```
usage: ./SSHatter.pl <maximumprocess> <targetserverfilename> <usernamefilename>
<passwordfilename> <sleeptime> <timingflag>
sleeptime: 0 - disable retries at ./SSHatter.pl line 62.
```

```
mideb:/home/ssh/SSHatter-0.6/src# ./SSHatter.pl 1 ./servers ./users ./passwords 5 1
[sshatter] trying testuser@192.168.111.22:22/1234
[sshatter] testuser@192.168.111.22:22:38.5799 seconds elapsed
[sshatter] trying testuser@192.168.111.22:22/password
[sshatter] testuser@192.168.111.22:22:38.1768 seconds elapsed
[sshatter] trying testuser@192.168.111.22:22/foobar
[sshatter] testuser@192.168.111.22:22:40.4126 seconds elapsed
```

```
Oct 8 20:07:35 localhost sshd[10577]: fatal: Timeout before authentication
for ::ffff:192.168.111.22
Oct 8 20:07:35 localhost sshd[10578]: fatal: Timeout before authentication
for ::ffff:192.168.111.22
Oct 8 20:07:35 localhost sshd[10579]: fatal: Timeout before authentication
for ::ffff:192.168.111.22
```

**FAZIT:** Ich habe keine SSH Verbindung zustande gebracht – ich nehme an, dass dies mit einer inkompatiblen Perl Version oder einer zu neuen SSH Server Version zusammenhängt.

## THC Hydra v5.4

Ein Multiprotokoll Cracker, dieses Tool kann folgende Protokolle angreifen:

*Currently this tool supports:*

*TELNET, FTP, HTTP, HTTPS, HTTP-PROXY, SMB, SMBNT, MS-SQL, MYSQL, REXEC, RSH, RLOGIN, CVS, SNMP, SMTP-AUTH, SOCKS5, VNC, POP3, IMAP, NNTP, PCNFS, ICQ, SAP/R3, LDAP2, LDAP3, Postgres, Teamspeak, Cisco auth, Cisco enable, LDAP2, Cisco AAA (incorporated in telnet module).*

Info: zum kompilieren muss libssh v0.11 installiert sein, URL: <http://0xbadc0de.be/libssh/libssh-0.11.tgz>.  
(Version 0.20 funktioniert nicht!)

```
fwpiercer:/tmp/hydra-5.4-src# ./hydra -v -l root -P pw localhost ssh2
[VERBOSE] More tasks defined than login/pass pairs exist. Tasks reduced to 7.
Hydra v5.4 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2007-12-29 19:30:41
[DATA] 7 tasks, 1 servers, 7 login tries (l:1/p:7), ~1 tries per task
[DATA] attacking service ssh2 on port 22
[VERBOSE] Resolving addresses ... done
[STATUS] attack finished for localhost (waiting for childs to finish)
[22][ssh2] host: 127.0.0.1 login: root password: password
Hydra (http://www.thc.org) finished at 2007-12-29 19:30:46
```

**FAZIT:** Ein interessantes und funktionierendes Tool, welches nicht nur auf SSH2 beschränkt ist.

## ScanSSH v2.0

Vorhanden als Debian Paket. Dieses Tool ermöglicht ganze Subnetze nach vorhandenen SSH Server (und weiteren Diensten wie SOCKS, http Proxy...) zu durchsuchen:

```
# scanssh -s ssh,http-connect 80.219.160.0/24
80.219.160.42:22 SSH-2.0-Cisco-1.25
```

Ich habe ein paar Subnetze gescannt, hier eine nicht repräsentative Auswahl von SSH Versionen im Einsatz:

```
80.219.160.225:22 SSH-2.0-dropbear_0.48
80.219.162.155:22 SSH-1.99-OpenSSH_4.5
80.219.159.127:22 SSH-2.0-OpenSSH_4.3
80.219.159.29:22 SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
80.219.158.7:22 SSH-2.0-OpenSSH_4.1p1 Debian-7ubuntu4.2
80.219.158.184:22 SSH-1.99-OpenSSH_4.5
80.219.158.233:22 SSH-2.0-Sun_SSH_1.1

80.74.145.237:22 SSH-1.99-OpenSSH_3.6.1p2
80.74.145.242:22 SSH-1.99-OpenSSH_3.6.1p2
80.74.145.230:22 SSH-2.0-OpenSSH_3.8.1p1 FreeBSD-20040419
80.74.145.231:22 SSH-2.0-OpenSSH_3.9p1
80.74.145.228:22 SSH-2.0-OpenSSH_3.8.1p1 FreeBSD-20040419
80.74.145.229:22 SSH-2.0-OpenSSH_3.8.1p1 FreeBSD-20040419
80.74.145.227:22 SSH-2.0-OpenSSH_3.8.1p1 FreeBSD-20040419

80.74.147.3:22 SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
80.74.147.6:22 SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
80.74.147.10:22 SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
80.74.147.2:22 SSH-2.0-OpenSSH_3.8.1p1 Debian-8.sarge.4
80.74.147.16:22 SSH-2.0-OpenSSH_4.3p2 Debian-9

80.74.146.65:22 SSH-1.99-OpenSSH_3.9p1
80.74.146.181:22 SSH-1.99-OpenSSH_3.6.1p2
80.74.146.219:22 SSH-2.0-OpenSSH_4.3p2 Debian-9

Effective host scan rate: 7.91 hosts/s
```

**FAZIT:** relativ schnelles (ca. 8 Hosts/s) und brauchbares Tool um SSH Server zu finden.

## SSH-Privkey-Crack

Ich habe kein private key Brute-Force Tool gefunden, das unter Windows läuft. Eine alte Linux Version eines Brute-Force Tools habe ich gefunden, jedoch war die Performance sehr schlecht. Ich habe das Tool angepasst, dass es auch mit Windows Textfiles umgehen kann und die Performance erhöht.

Man kann den private key entweder via Wordlist oder mit „John the Ripper“<sup>36</sup> „angreifen“. Das Tool testet ca. 16'500 Keys pro Sekunde (natürlich Hardware abhängig), also nicht wirklich performant.

Ein Beispiel um den Passphrase mit einer Wordlist zu finden:

```
mideb:/home/ssh/john-1.7.2/run# ./ssh-privkey-crack032 id_dsa < password.lst
ssh-privkey-crack032 v0.3 made by anonymous@echo.or.id, enhanced by michu@neophob.com
keyheader:
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC,3AD7499647AD2095

trying inf keys/s, # of tested keys: 3001.
-----
Passphrase match: <password>. Found password after 0 seconds and 3112 tries.
-----
```

Hier ein Beispiel, um den Passphrase „password“ mit „John the Ripper“ zu cracken (ein schlechter Passphrase!):

```
# ./john -stdout -incremental | ./ssh-privkey-crack id_dsa
ssh-privkey-crack032 v0.3 made by anonymous@echo.or.id, enhanced by michu@neophob.com
keyheader:
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC,3AD7499647AD2095
trying 15889 keys/s, # of tested keys: xxxx
-----
Passphrase match: <password>. Found password after 19433 seconds and 291195045 tries.
-----
```

Nach 5 Stunden 24 Minuten und 291 Millionen versuchten Passwörter war dieser einfache und daher schlechte Passphrase auf einem P4 Rechner gecrackt.

Ich habe einen weiteren Versuch unternommen, diesmal hatte der private key ein Passwort von P@ssOrd. Aufgrund des erweiterten Alphabets habe ich den Versuch nach knapp 2,5 Milliarden Versuchen und nach knapp 3 Tagen aufgegeben.

**FAZIT:** Ist der Passphrase nicht vorhanden oder muss ein Passwort eines Keys wiedergefunden werden, ist Geduld, Glück und Rechenpower nötig.

Eine interessante Idee: „BackTrack John The Ripper MPI Instant Cluster“<sup>37</sup> mittels Backtrack Live CD mehrere Rechner zu einem Cluster zusammenschliessen, welche dann das Passwort gemeinsam cracken.

<sup>36</sup> <http://www.openwall.com/john/>

<sup>37</sup> <http://offensive-security.com/documentation/backtrack-cluster.pdf>

## PuTTY Private Key Crack

Ich habe den PuTTY-Plink Source abgeändert um PuTTY generierte private keys mit der Brute Force Methode zu cracken. Dieses Tool arbeitet gut mit "John the Ripper" zusammen, eines der besten Tools für Brute Force Angriffe. Hier ein Beispiel

```
Microsoft windows [Version 6.0.6000]
Copyright (C) 2006 Microsoft Corporation. Alle Rechte vorbehalten.
C:\>john -stdout -incremental | p-ppk-crack password.ppk
p-ppk-crack v0.5 made by michu@neophob.com - PuTTY private key cracker
len: 148/352
trying 65808 keys/s, # of tested keys: 1711001.
-----
Passphrase match: <password>. Found password after 26 seconds and 1711001 tries.
-----
```

Eine interessante Beobachtung; unter Windows 2000 berechnete p-ppk-crack 36310 keys/s. Auf diesem Rechner lief ein Window Vista in einer VMWare Workstation. In dieser virtuellen Maschine berechnete das Tool 65808 keys/s! Der physikalische CPU ist ein AMD Athlon 64 X2 Dual Core 5200+.

**FAZIT:** Dieses Tool sollte aufzeigen, dass wirklich einen PASSPHRASE und kein PASSWORT verwenden werden sollte, also ein ganzer Satz und nicht nur ein Wort.

## SSHUTOUT v1.0.5

Dieses Tool überwacht das SSH Logfile, bei einem Brute Force Angriff auf den SSH Server wird dieser Daemon aktiv und schliesst den SSH Port (via iptables) für eine definierte Zeit.

Mit dem Tool Hydra (siehe oben) habe ich einen Brute Force Angriff simuliert:

```
fwpiercer:~/hydra-5.4-src# ./hydra -v -l root -P pw 192.168.238.129 ssh2
Hydra v5.4 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2008-01-22 10:11:26
[DATA] 12 tasks, 1 servers, 12 login tries (1:1/p:12), ~1 tries per task
[DATA] attacking service ssh2 on port 22
[VERBOSE] Resolving addresses ... done
[STATUS] attack finished for 192.168.238.129 (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2008-01-22 10:11:28
```

Sshutout wurde mit dem Parameter „-D“ gestartet, damit der Output angezeigt wird:

```
fwpiercer:~# ./sshutout-1.0.5/sshutout -D
sshutout ver. 1.0.5 -- (C)Copyright 2007 - Bill DuPree - All rights reserved.
*** The sshutout 1.0.5 daemon has started ***
sshutout configuration follows:
Configuration file: /etc/sshutout.conf
SSH Daemon: sshd
Input log file: /var/log/auth.log
Output log file: /var/log/sshutout.log
PID file: /var/run/sshutout.pid
Polling interval: 60 seconds
Threshold: 4 attempts
Delay penalty: 300 seconds
Portscan squelching is disabled
Illegal/Invalid user squelching is disabled
whitelist:
  None
Squelching attack from 192.168.238.129 (10 ssh login attempts) for 300 seconds.
Unblocking 192.168.238.129 after expiry of 300 seconds.
```

Hier eine ungetestete IPTables rule, welche ähnliches leisten sollte:

```
iptables -t nat -A prerouting_rule -p tcp --dport 22 -m state --state NEW -m recent --name
ATTACKER_SSH --rsource --update --seconds 180 --hitcount 3 -j DROP && iptables -t nat -A
prerouting_rule -p tcp --dport 22 -m state --state NEW -m recent --name ATTACKER_SSH --rsource
--set
```

**FAZIT:** Kennt man sich nicht wirklich mit iptables aus, ist dieses Tool sicherlich zu empfehlen, da es einfach in der Handhabung ist und funktioniert. Jedoch sollte mit iptables das Gleiche möglich sein – ohne zusätzlichen Daemon.

## WEITERE TOOLS IM ZUSAMMENHANG MIT SSH

### Port-Knocking

Port Knocking ist eine Technik, welche vor allem automatisierte Tasks (Portscanning, Würmer, etc.) abhalten soll, Informationen über unser System zu erhalten.

Port Knocking ist eine zusätzliche Schicht, welche die Firewall-Schicht umgibt. Alle Ports in diesem Beispiel sind geschlossen, ein Port-Scan wird keine Ergebnisse aufzeigen. Der User schickt nun eine bestimmte „Port-Knock“ Sequenz, an den Server. War es die korrekte Sequenz öffnet der Port Knock Daemon den entsprechenden Port an der Firewall (mit iptables) und die SSH Verbindung kann aufgebaut werden. Das Ganze grafisch dargestellt<sup>38</sup>:

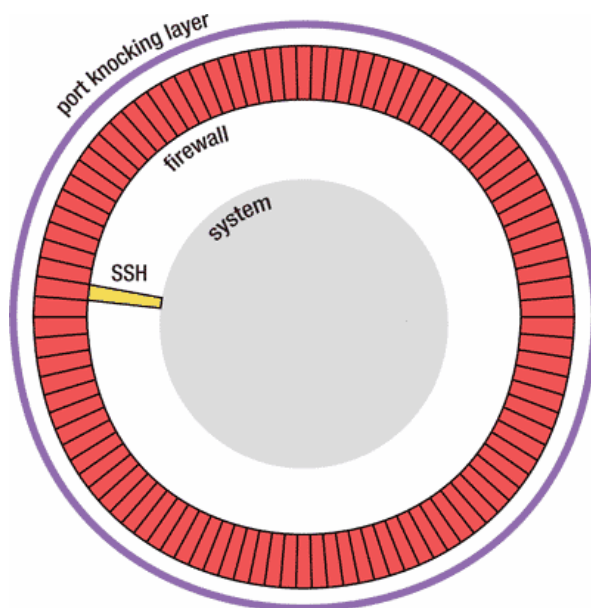


Abbildung 33: Port Knocking, normaler Zustand

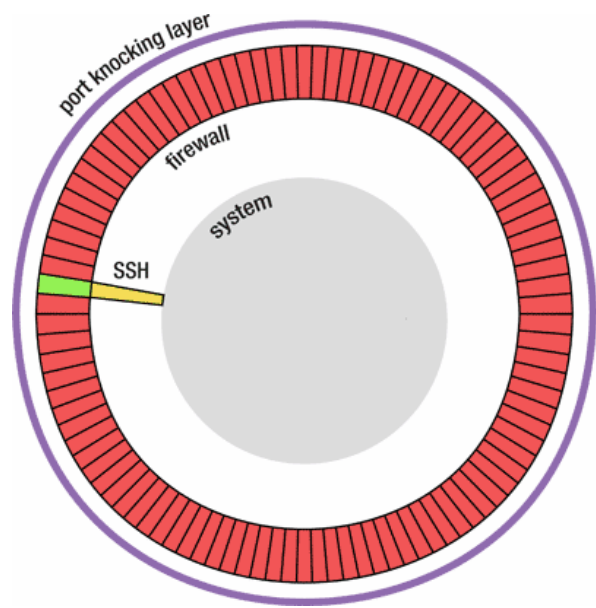


Abbildung 34: Port Knocking, die Firewall nach einer erfolgreichen Port-Knock-Sequenz

Damit dies funktionieren kann (d.h. bevor die Anfragen verworfen werden), muss der Daemon im Link Layer arbeiten (2te Schicht im OSI Model) oder aber die Firewall Software muss ein Logfile führen, indem aufgeführt wird, was verworfen wurde.

Praktisch habe ich port knocking nicht getestet. In meinem Fall macht es keinen Sinn, da ich meinen SSH Server als Gateway verwenden möchte. Muss man nun vor jeder Verbindung zuerst noch den Port mit dem knock client öffnen, ist das wieder eine Fehlerquelle mehr (siehe Kapitel Bypass Proxy und Bypass Firewall).

Bei einem System, das fernwartet wird, wäre das sicherlich eine gute Wahl.

<sup>38</sup> Bilderquelle: <http://www.portknocking.org/view/about/critique>

## SSH BEST PRACTICES

Basierend auf den bekannten SSH Attacken können folgende Aussage über „SSH best practices“ gemacht werden:



- Nur SSH Version 2 verwenden (fallback auf Version 1 deaktivieren)
- Root login verbieten. Das verringert die Brute-Force Erfolgchance, da zuerst ein gültiger User gefunden werden muss
- Aktueller SSH Client und Server verwenden
- Ist ein Passwort basierendes Login erlaubt, unbedingt komplexe Passwörter verwenden (siehe Brute Force Tools im letzten Artikel)
- Werden public keys verwendet, diese zwingend mit einem Passphrase schützen.
- Die ForwardAgent Option (Server) sollte nur eingesetzt werden, wenn die Personen, welche root Rechte auf dem Server haben, vertraute Personen sind.
- Der SSH Port des Servers von Standard Port 22 auf einen anderen Port wechseln. Das schützt wenigstens vor simplen Scanversuchen.

Obwohl ich keine erfolgreiche Attacke gegen meinen SSH Server durchführen konnte (Brute Force zähle ich nicht als technische Attacke) heisst das natürlich nicht, dass es keine Attacken gibt. Auf meinem Public SSH Server konnte ich relativ viele Brute Force Attacken feststellen, diese „Gefahr“ ist also durchaus real und wahrscheinlich die häufigste Angriffsmethode gegen SSH Server.

PuTTY (und andere SSH Produkte) hatte eine Sicherheitslücke bis und mit Version 0.53, nachzulesen unter „CERT Advisory CA-2002-36 Multiple Vulnerabilities in SSH Implementations“<sup>39</sup>.

Fazit: Das schwächste Glied in der Kette im Zusammenhang mit SSH2 dürfte meistens der Mensch, respektive der Administrator sein: schlechte Konfiguration oder einfach zu erratene Passwörter.

<sup>39</sup> <http://packetstorm.dtecks.net/advisories/cert/CA-2002-36.ssh>



## REFERENZEN

### BILDREFERENZEN

Abbildung 1: SSH Subsystems.....	5
Abbildung 2: OpenSSH Logo .....	6
Abbildung 3: PuTTY Konfiguration .....	7
Abbildung 4: PuTTY in Action.....	7
Abbildung 5: WinSCP in Action.....	8
Abbildung 6: invalider SSH fingerprint .....	10
Abbildung 7: Public Key Vorgang.....	11
Abbildung 8: Übersicht der Laborumgebung.....	14
Abbildung 9: PuTTYGEN .....	14
Abbildung 10: Einer PuTTY Session wird ein Private Key zugewiesen .....	15
Abbildung 11: SSH Agent Verwendung .....	17
Abbildung 12: PuTTY Agent Weiterleitung .....	17
Abbildung 13: Netzwerk Übersicht.....	20
Abbildung 14: PuTTY, Verbindung herstellen .....	20
Abbildung 15: PuTTY, Tunnel erstellen .....	20
Abbildung 16: Eine RDP-over-SSH Verbindung wird aufgebaut.....	20
Abbildung 17: Loopback NIC installieren .....	21
Abbildung 18: Konfiguration der Loopback NIC .....	21
Abbildung 19: Lokaler SSH Tunnel.....	22
Abbildung 20: Ein Client verbindet sich zu einem Internet Server.....	23
Abbildung 21: Ein Server verbindet sich via SOCKS Proxy zu einem Internet Server .....	23
Abbildung 22: PuTTY als SOCKS Proxy .....	24
Abbildung 23: SOCKSv1 .....	24
Abbildung 24: Netzwerk Übersicht Remote Tunnel .....	26
Abbildung 25: Firmen Workstation, SSH Tunnel Einstellung.....	27
Abbildung 26: Home Workstation, SSH Tunnel Einstellung.....	27
Abbildung 27: MS RDP Client .....	27
Abbildung 28: PuTTY Proxyeinstellungen .....	32
Abbildung 29: Verbindungsaufbau via HTTPtunnel.....	35
Abbildung 30: Verbindungsaufbau via HTTPtunnel und NTLMs .....	36
Abbildung 31: Funktionsweise von ptunnel.....	38
Abbildung 32: Verbindungsaufbau via ptunnel .....	39
Abbildung 33: Port Knocking, normal Zustand .....	45
Abbildung 34: Port Knocking, die Firewall nach einem erfolgreichen Port Knock Sequenz .....	45

Die verwendeten Illustrationen sind unter <http://www.culture.gouv.fr/culture/atp/cdrom/anglais/cfv8.htm> zu finden.