

Tutorials Software Anleitungen (HowTos) rund um Mac-Software

WebSite-Analyse in Echtzeit

Nie wieder Ungewißheit durch Logfiles!

apfeltalk wird unterstützt von webEdition

page **L**ogger




[antworten](#)

[Themen-Optionen](#) ▾

[Thema durchsuchen](#) ▾

[Bewertung:](#) ■■■■■ ▾

[Ansicht](#) ▾

 Gestern, 13:15

#1

Cyrics

Fuji



Registriert seit: 04.2005

Ort: Leipzig

Alter: 21

Beiträge: 1.299



[\[OS X\]SSH-Remote-Lösungen unter OSX \(dazu VNC, KVM\)](#)

Hey,

da manche auf Remote-Lösungen angewiesen sind, möchte ich ein kleines Tutorial für SSH, VNC und ein paar Einblicke in KVM over IP anbieten. Ich glaube nur die Wenigsten werden hierfür Verwendung finden, aber wenn sich ein paar Wenige finden, die ich damit glücklich machen kann, dann hat es ja schon etwas gebracht.

Vorweg sei noch gesagt, dass ich etwas sehr ins Detail gehen werde um auch denjenigen, die bisher keine Ahnung von der Materie haben das entsprechende Grundwissen vermitteln zu können. Für diejenigen, die sowieso schon alles näher wissen und nur 1-2 Befehle suchen, wird es hier wohl nicht das Richtige sein.

WARNUNG:

SSH ist ein ziemlich sicheres Protokoll. Es kann jedoch keine Schwachstellen im zugrunde liegenden System verhindern. Außerdem gibt es von SSH zwei existierende Protokolle. Das SSH1-Protokoll ist veraltet und weist Schwachstellen auf, die im SSH2 geschlossen wurden. Wenn ein Fehler im Grundsystem oder gar im OpenSSH liegt, hilft auch keine sichere Verbindung. Möglicherweise ist dann der SSH-Server so offen wie ein Windows-Rechner oder das sprichwörtliche Scheunentor. Jeder ist für die Sicherheit seines Systems verantwortlich! Ich werde Tipps geben wie man seinen SSH-Server so sicher wie möglich machen kann. Es kann aber auch für manche soweit gehen, dass sie gar keinen Remote-Zugriff mehr auf ihren SSH-Server besitzen. Um dies zu vermeiden, ist es anfangs zwingend erforderlich, dass sowohl der SSH-Server als auch der SSH-Client physikalisch erreicht werden können. Ich bin gerne behilflich bei allen Fragen, und eigentlich ist es oft besser vorher zu fragen als es auszuprobieren, gerade bei einer solchen Remote-Lösung, aber ich distanziere mich trotzdem ausdrücklich von eventuell entstehenden Schäden, falls jemand seine Daten plötzlich im Internet verstreut findet.

Zur Struktur des Tutorials:

1. Remote - Erklärungen, Definitionen
2. Voraussetzungen (Netzwerk, DMZ, ...)
3. SSH
 1. SSH - Allgemein
 2. SSH-Server manuell einrichten
 1. Server vorbereiten
 2. Privat- und PublicKey am Client erzeugen
 3. Übermittlung der Schlüsseldatei
 4. manuelle SSH-Konfiguration des Servers
 5. abschließende optionale SSH-Konfiguration des Clients
 3. Per SSH Anwendungen tunneln (Port-Forwarding)
 4. SSH-Server schnell-konfiguriert durch SSH-Helper
4. DynamicDNS
 1. Allgemein Dynamic DNS und dessen Vorteile

2. Dynamic DNS - Account einrichten
3. Router für DynDNS konfigurieren
4. Server mit DynDNS-Dienst einrichten
5. die nötige Router-Konfigurierung
6. VNC
 1. Vergleich SSH und VNC
 2. VNC - Server
 3. VNC - Client
 4. VNC - Optimierung
7. Alternativen: KVM-over IP-Lösung
8. Abschließende Worte

1. Remote - Erklärungen, Definitionen

Eine Remote-Lösung heisst im engeren Sinne die Fernbedienung eines Objektes. Hierbei konzentriere ich mich aber nur auf die Fernsteuerung eines Computers (hier nur Macintoshs und eigentlich alle Unix-Systeme) durch einen anderen Computer. Es ist das Ziel den Computer über eine sichere Verbindung so fernzusteuern, dass kein physikalischer Eingriff mehr an dem zu administrierenden PC geschehen muss. Voraussetzung ist hier, dass zumindest eine Netzwerkanbindung vorhanden und Netzanschluss mit dem Computer verbunden ist und die grössten Einstellungen vorgenommen wurden. D.h. es wurde das Betriebssystem installiert und eine erfolgreiche Verbindung zum Internet hergestellt. Es muss mindestens ein User (am besten zwei User) angelegt werden. (Admin-Anwender-Sicherheitskonstruktion). Der Anwender kann und darf kein FileVault-Image nutzen! Denn er wird unser Login für den SSH-Zugriff. Bei FileVault kommt der SSH-Server nicht an die benötigten Dateien. Denn es kann keine Verbindung aufgebaut werden, da das FileVault-Image erst nach dem Login geöffnet wird, und der SSH-Server somit nicht an die nötigen Einstellungen des unseres Login-Benutzers gelangt. Der vorgeschaltete Router muss umkonfiguriert werden und darauf werd ich auch noch näher eingehen. Die Remote-Lösung muss auf dem Rechner installiert und einsatzbereit sein.

Das wäre im näheren:

Für OpenSSH ist bereits auf OSX- und jedem Unix-System vorinstalliert. Für eine praktikablen Handhabung mit SSH gibt es ein nützliches kostenloses Werkzeug:

Für die einfachste SSH-Lösung (kein manuellen Eingriff nötig)

[SSH-Helper](#)

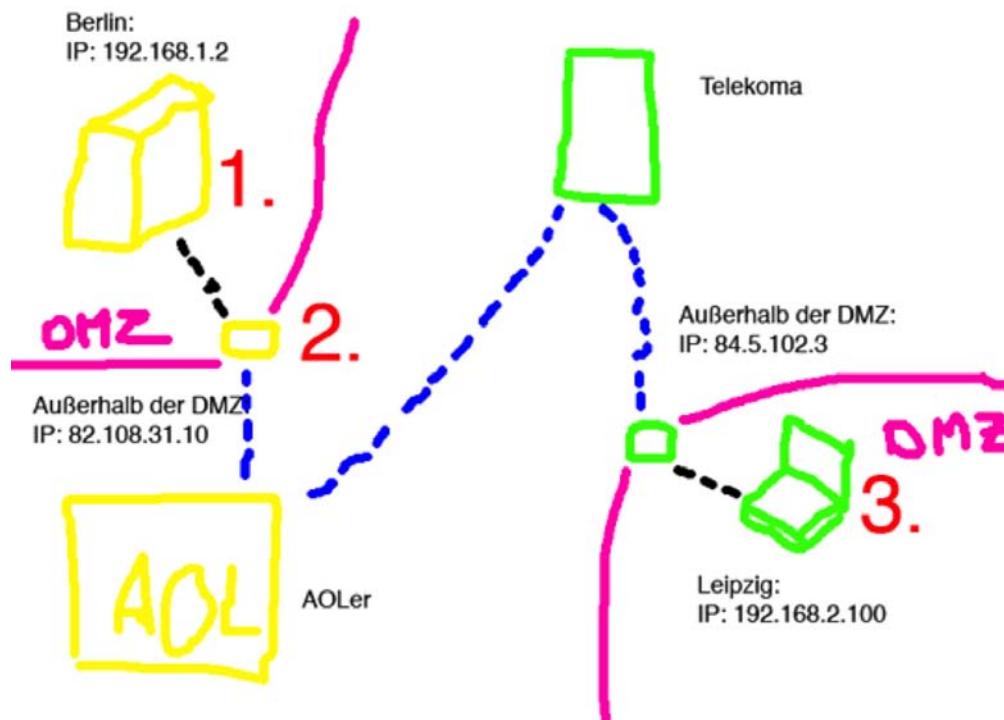
Für die VNC-Remote-Lösung:

[VNC-Server](#)

[VNC-Viewer](#)

Nun erklär ich "schnell" die Struktur des Ganzen. Damit auch Verständnis für die ganzen Änderungen, die vorgenommen werden müssen aufkommt. Denn wenn man keine Ahnung von dem hat, was man tut, macht man oft die gravierendsten Fehler.

Als Grafik hab ich da etwas schnell hingemalt... im wahrsten Sinne des Wortes. Ich bitte um Verständnis, aber so eine einfache Zeichnung verdeutlicht den eigentlichen Gedanken oft am besten.



2. Von Grund auf... Netzwerke, DMZ und so...

Wir haben unseren Server in Berlin stehen und den Client, also den Rechner mit dem wir auf dem Server zugreifen wollen, in Leipzig, bzw. die Konstruktion streben wir als Ziel an. Anfangs sind beide im gleichen Netzwerk meinetwegen in Berlin. Bei der Ziel-Vorstellung haben Beide unterschiedliche Internet-Anschlüsse und damit unterschiedliche Internet-Provider (nein, ich mach hier keine Werbung 😊). Und beide Netzwerke haben natürlich jeweils einen anderen Router. Der Router übernimmt folgende Funktionen im privaten Netzwerk: er gibt jedem PC, der sich im privaten Netzwerk anmeldet eine IP-Nummer, die den Rechner konkret und einzigartig für das private Netzwerk ausweist. Bei mir in Leipzig wäre für meinen Client die IP-Nummer 192.168.2.100. Dies ist eine **private Netzwerk-IP-Nummer** und damit ein Klasse C Netz. Das gleiche ist bei dem Netzwerk in Berlin. Dort hat unser Server die IP-Nummer 192.168.1.2. Der Router hat eine zweite Funktion. Er verbindet das private Netzwerk Klasse C mit dem außenstehenden Netzwerk, dem Internet. Daher bildet er die Barriere zwischen Internet und Klasse C-Netz. Er bildet somit die **demimilitarisierte Zone (DMZ)**. Den Schutz und Abgrenzung eines privaten Netzwerkes zu anderen Klassen-Netzwerken. Klasse C Netzwerke haben die Subnetzmaske 255.255.255.0. D.h. das Netzwerk ist nur auf der letzten Position variabel bis maximal 254 Rechner. Der 255. muss ja der Router selbst sein. In der Klasse C kann man sich eine beliebige IP-Nummer auswählen, je nachdem ob der Router die freie Wahl dieser zulässt. Aber da die Klasse C eben auf bestimmte Nummern festgelegt ist, ist meistens auch der Router auf diese Nummern festgesetzt. Bei der externen IP-Nummer erhält man dagegen eine IP-Adresse, die vom Internet-Anbieter festgelegt ist. Dieser hat auch nur ein paar Nummern "gekauft" und man bekommt meistens "fast" die gleiche. Es bleibt immer in einem bestimmten Abstand, wenn man ein bisschen darauf achtet. Seine Ausgangs-IP erfährt man am einfachsten z.B. durch Webseiten wie whatismyip.de. Dort wird einfach nur die IP-Nummer ausgegeben mit der man auf die Webseite zugreift. Mit dieser externen IP-Nummer, die sich mindestens 1mal alle 24 Stunden ändern muss (wird durch den Internet-Anbieter zwangsgetrennt) ist jeder User eindeutig identifizierbar. Daher ist es selten angebracht seine Ausgangs-IP in Chats und dergleichen zu veröffentlichen! Wenn man in einem anderen Netzwerk (beispielsweise auf Arbeit) ist und zu seinem Netzwerk zuhause zugreifen will, dann braucht man bloss die Ausgangs-IP des Routers daheim. Je nachdem wie dieser konfiguriert ist und welche Dienste laufen kann man dann etwas an Infos erhalten oder komplett abgeblockt werden. In diesem Tutorial geht es darum von seinem Netzwerk in Leipzig auf das Netzwerk in Berlin zuzugreifen, und der Router im Berliner Netzwerk muss auch soweit konfiguriert werden, dass er Zugriffe von außen auf den Server durchlässt. Also haben wir etwas vor uns...

3. SSH

3.1. SSH - Allgemein

Ich beziehe mich hier nur auf OSX-Systeme, aber es handelt sich wie gesagt um den Dienst OpenSSH und das Protokoll SSH2, und dies ist in vielen BSD-Unix-Systemen, wie auch in Debian eingebunden. Es ist auf fast alle Unix-Systeme übertragbar.

Eine SSH-Verbindung kann per Passwort oder mit einem vorgefertigten Zugriffsschlüssel oder beidem geschützt werden. Passwörter über das Netzwerk über sonst wieviele Knotenpunkte zu verschicken bedeutet aber immer ein Sicherheitsrisiko. Außerdem besteht die Problematik, dass Passwörter geknackt werden können durch ein einfaches Skript, welches alle Standard-User und alle Standard-Passwörter durchprobiert. Früher hatte ich etwa 120 SSH-Zugriffe in der Nacht auf meinem Server. Die **Skript-Kiddies** waren da sehr aktiv und ich mit meinen **Gegenmaßnahmen** auch.

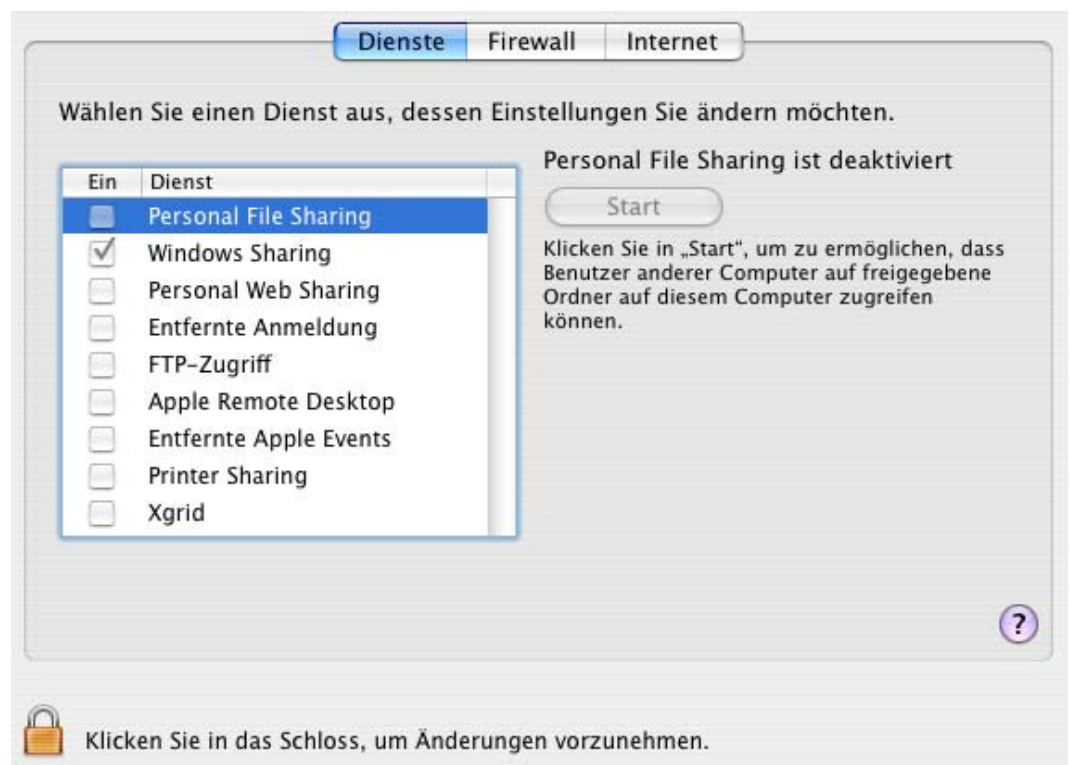
Ich werde hier den Weg der sicheren SSH-Authentifizierung erklären. Dabei identifiziert man seinen Client, mit dem man auf den Server zugreift, mit einer Schlüsseldatei (**Keyfile**). Diese hat eine von uns gewünschte Verschlüsselung mit einer von uns gewünschten Bitgröße. Durch diese verhindern wir Attacken wie dem bekannten Problem des **Man-in-the-Middle-Angriffes** (MITM). Wir brauchen uns dann nicht mehr per Passwort und User-ID identifizieren, da dies alles in der Keyfile enthalten sein wird. Dies lässt einmal die Sicherheitslücke des übertragenen Passwortes über das Internet verschwinden, aber andererseits auch die Hackerskript-Versuche, die sehr oft auf eine Passwort-Authentifizierung setzen. Zusätzlich zur Keyfile können wir ein optionales Passwort setzen. Dies nennt sich im Zusammenhang mit der Keyfile **Passphrase**, welches die eigentliche Verschlüsselung der Schlüsseldateien bildet. Dazu später mehr.

TIPP: im Terminal `man ssh` eintippen um sich das Manual zu SSH durchzulesen, und `man sshd` um zusätzliche Informationen über den SSH-Daemon zu erhalten, der den SSH-Server bildet. ssh:

- **-l username:** l steht für Login, und übermittelt mit der Anfrage den Benutzernamen, mit dem wir versuchen uns auf dem Server einzuloggen.
- **-i /ort/der/identity:** i steht für identification und mit diesem Parameter kann man einen anderen Ort als den standardmässigen `~/.ssh/id_rsa` angeben.
- **-p Port:** Hier kann ein anderer Port optional angegeben werden, falls der SSH-Server auf einen anderen Port auf Anfragen lauscht.
- **-F /ort/der/sshd_config-File:** hiermit lässt sich eine andere `sshd_config`-Datei von unserem Client nutzen.
- **-L Startport:Hostname:Zielport:** sehr nützlich für IP-Forwarding per SSH

Auf dem Server:

Wir gehen in die **Systemeinstellungen** -> **Sharing** -> **Dienste** und aktivieren den Dienst **Entfernte Anmeldung**. Damit wird der Remote-Zugriff per SSH gestattet.



Anfangs sollten beide Computer (Client und Server) in ein und dem selben Netzwerk sein um die SSH-Verbindung testen zu können. Wenn der Dienst aktiviert ist, passt die Firewall die Einstellungen automatisch an, falls diese aktiviert ist. Es wird Port 22 geöffnet für Zugriffe und der SSH-Server gestartet, und damit können wir uns vom Client auf dem Server einloggen, wenn

er im selben Netzwerk ist.

Dazu öffnen wir das Terminal (Dienstprogramme) und tippen dort ein einfaches

Code:

```
ssh username@serverip
```

oder das gleich bedeutende

Code:

```
ssh -l username serverip
```

ein. Man kann sich auch zu sich selbst per SSH verbinden indem man die *serverip* mit *localhost* ersetzt.

Der *username* ist selbstverständlich unser Login-User am Server. Die *serverip* lässt sich beim Server erfahren bei *Systemeinstellungen -> Netzwerk -> aktive Netzwerkschnittstelle -> TCP/IP*. Oder in der *Menüleiste Apfel-> Über diesen Mac -> weitere Informationen -> Netzwerk -> IPv4-IP-Adresse*. Dort steht die aktuell durch den Router zugeteilte IP-Nummer.

Höchstwahrscheinlich wird noch nicht viel passieren außer das die Computer sich nun gegenseitig versuchen zu identifizieren. Es werden MAC-Adresse etc. ausgetauscht. Es wird ein einzigartiger Finger-Print am Client, mit dem wir uns auf den SSH-Server verbinden, angezeigt. Dieser Fingerprint informiert uns darüber, ob wir uns auf den richtigen Server verbinden. Dies kennt man vielleicht von Bank-Servern etc., die ihre Fingerprints bei Internet-Banking aufzeigen, damit man sich sicher sein kann auf dem richtigen Bank-Server seine Daten einzugeben.

Wir bestätigen den Server-Fingerprint mit einem yes bei der Nachfrage.

3.2 SSH-Server manuell einrichten

3.2.1 Server vorbereiten

Wir befinden uns also am Server:

Dort öffnen wir das *Terminal*. Dieses liegt standardmässig im Ordner *Dienstprogramme*.

Wenn wir das Terminal öffnen, landen wir wie gewöhnlich im User-Verzeichnis.

Hier sollten wir als erstes ein

Code:

```
mkdir .ssh
```

eintippen. Falls noch kein solches Verzeichnis bei uns im User-Verzeichnis, welches unser Login auf dem Server sein wird, existiert.

3.2.2 Privat- und PublicKey am Client erzeugen

Wir sind am Client:

Nun wollen wir unsere SSH-Keyfile erzeugen. Bei der SSH-Verschlüsselung liegt bei jedem Unix-System das nützliche kleine Werkzeug *ssh-keygen* bei. SSH erkennt zwei Algorithmen als Verschlüsselung an, das ist einmal der *RSA*- und zum anderen der *DSA-Algorithmus*.

Welches von beiden besser ist... das ist schwierig. Niemand legt sich so recht fest. RSA-Keys sind länger als DSA. Dadurch angeblich leichter zu knacken als DSA-Keys, weil man bei RSA mehr Daten zur Verfügung hat mit den man handeln kann. DSA-Keys brauchen dafür ihre Zeit bis man sich mit diesen am Server verifiziert hat. DSA-Keys sollen hauptsächlich nur Daten-Signaturen sein mit der man sich identifiziert. Bei RSA fließt durch das Passwort als auch die Signature eine andere Verschlüsselung ein. Ich will mich da selbst nicht festlegen. DSA ist auf jeden Fall nur für SSH2-Server. RSA dagegen ist zweigeteilt in RSA1 und RSA. RSA ist sowohl für SSH1 als auch für SSH2. RSA1 ist nur für SSH1-Server. Man sollte unbedingt RSA1 bzw. das Protokoll SSH1 vermeiden!

In meinem Beispiel werde ich die RSA-Algorithmierung nutzen. Diese lässt sich aber leicht abwandeln zu DSA.

Um einen Schlüssel zu generieren tippen wir ins Terminal folgendes ein:

Code:

```
ssh-keygen -b 2048 -t -rsa
```

Mit *-b* legen wir die Bitstärke fest. 1024 Bits sind da Standard. Ich lege gerne doppelt soviel drauf. Damit ist der entstehende Schlüssel 2048 Bits gross (was 2 KB entspricht). Mit *-t* legen wir

den Typen der Algorithmierung fest. Ich hab mich für RSA entschieden. Was aber wie gesagt leicht durch DSA ausgetauscht werden kann.
 Die Erstellung eines RSA-Keys sollte seine Zeit dauern (ein paar Sekunden). DSA-Schlüssel sind dagegen schneller erstellt.
 Nachdem der Schlüssel berechnet wurde, werden wir gefragt, ob wir den Schlüssel in dem vorgeschlagenen Verzeichnis (`~/.ssh/id_rsa`) speichern wollen. Und das wollen wir, außer dort lagern bereits SSH-Schlüssel für andere Server. Dann sollte man natürlich die Standard-Bezeichnung abändern, indem man den Pfad abtippt und den Schlüsselnamen ändert.
 Wir werden außerdem nun gefragt, ob wir einen Passphrase zusätzlich setzen wollen. Ein Passphrase ist eine optionale zusätzliche Identifikationsmethode. Man kann es sich vorstellen, dass der Schlüssel der Ausweis zum Club ist, aber man braucht zusätzlich wenn so gewünscht ein geheimes Passwort um Einlass zu finden. Ohne Passphrase reicht der Ausweis. Wenn wir aber den Ausweis verlieren, verlieren wir auch jeglichen Schutz!
 Technisch ausgedrückt ist der Passphrase die Verschlüsselung des Schlüssels. Nur derjenige, der den Passphrase kennt, kann den Schlüssel nutzen, bzw. ihn dechiffrieren. Ansonsten wird der Schlüssel zwischen Client und Server unverschlüsselt übermittelt. Dies bedeutet nur soweit ein Sicherheitsrisiko, wenn jemand unseren Netzwerk-Verkehr von außen versucht mitzuschneiden und aus den gefilterten Daten versucht den Schlüssel zu bekommen. Doch dazu später mehr, wie man dies verhindern kann.
 Diejenigen, die also ein Passphrase setzen wollen, tun dies jetzt. Die Bequemen, die ohne Passwort-Abfrage dann auf ihren Server zugreifen wollen, lassen die Eingabe hier frei mit zweimaligen Drücken von Enter.

Der Schlüssel wurde nun erstellt in dem angegebenen Verzeichnis und uns wird auch gleich das Erstellungsdatum sowie den Fingerprint des Schlüssels ausgegeben.

Nun zu näheren Erklärung:

wir haben nun zwei Dateien im `~/.ssh/` Verzeichnis. Standardmässig heisst die eine Datei `id_rsa` (oder `dsa` bei DSA-Verschlüsselung) und die andere `id_rsa.pub`.

`id_rsa` ist unser privater Schlüssel! Wenn wir diesen an Dritte verlieren, haben wir wirklich ein Problem! Es wie gesagt unser Ausweis, unsere Identifikation und sollte unter allen Umständen vor Dritten versteckt bleiben!

`id_rsa.pub` ist der öffentliche Teil unseres Schlüssels. Das Gegenstück sozusagen von unserer privaten Identifikation. `pub` ist wie man vielleicht erkennt die Abkürzung für `public`. Diese Datei wird bei den Servern platziert, wo wir uns identifizieren und um Einlass bitten wollen. Wir werden uns also nur mit unseren Ausweis dort einfinden, wo wir auch unser passendes Gegenstück finden. Somit ist hier der MITM-Angriff nicht mehr existent!

3.2.3 Übermittlung der Schlüsseldatei

Wir sind noch immer am Client:

Nun müssen wir die `id_rsa.pub` unserem Server übermitteln. Es ergeben sich verschiedene Möglichkeiten.

Die einfachste ist per SSH, welches wir vorhin ausprobierten um die Funktionalität zu testen, die Datei zu übermitteln.

Dies ist möglich mit folgender Befehlszeile:

Code:

```
scp ~/.ssh/id_rsa.pub username@serverip:~/.ssh/authorized_keys
```

Wir überschicken damit per sicherer Übermittlung die `id_rsa` an unseren Server in das Benutzerverzeichnis und benennen es gleichzeitig um zu der Datei `authorized_keys`. Dies hat sich einfach so eingebürgert mit der Bezeichnung, bzw. SSH-Server lassen sich soweit einschränken, dass sie nur auf Dateien, die `authorized_keys` heissen und die richtige Benutzer-Kennung besitzen, achten. Damit soll auch bewerkstelligt werden, dass weitere Clients auf den Server zugreifen können mit anderen Schlüssel. Jedoch wird mit dieser Methodik die `authorized_keys` überschrieben, falls vorhanden.

Aber wenn man per Netzwerk die Datei übermittelt und einlesen lässt, kann die `authorized_keys` mit Zugriffsschlüsseln sprichwörtlich aufgefüllt werden.

Dabei wird meinetwegen die `id_rsa.pub` in eine Freigabe kopiert, oder per `scp` übertragen mit einer anderen Ziel-Benennung als `authorized_keys`.

Auf dem Server sollte dann die `id_rsa.pub`, die in `authorized_keys` aufgenommen werden soll, im `.ssh`-Verzeichnis liegen.

Dann tippen ins Terminal:

Code:

```
cat id_rsa.pub >> authorized_keys
```

Dies funktioniert natürlich nur, wenn beide Dateien im selben Verzeichnis liegen. Ansonsten

muss vor `authorized_keys` noch der Ziel-Pfad angegeben werden, entweder relativ oder absolut.

iBook G4, 1.25Ghz, OSX 10.4, 80GB, 768MB

Debian PC-Server

iPod mini 2.G

Dell Widescreen 2005FPW

Geändert von Cyrics (Gestern um 20:39 Uhr).



Gestern, 13:16

2

Cyrics

Fuji



Registriert seit: 04.2005

Ort: Leipzig

Alter: 21

Beiträge: 1.299



3.2.4 manuelle SSH-Konfiguration des Servers

Nun sind die Schlüssel verteilt und wir passen die Einstellungen von SSH unseren Bedürfnissen an. Dabei wollen wir die unsichere Passwort-Authentifikation deaktivieren sowie auch gleich ein paar mehr Dinge für die Sicherheit unseres Servers tun.

Entweder wir loggen uns per SSH auf unseren Server ein (`ssh username@serverip`) oder wir sitzen direkt vor diesem.

Wir nutzen das Terminal und wechseln per `cd` das Verzeichnis zu `/etc`

Also:

Code:

```
cd /etc
```

Hier ist die Datei `sshd_config` abgelegt, welche die Einstellungen für unseren SSH-Dienst enthält.

Um diese zu öffnen nutzen wir am besten `pico` als Editor. Man kann aber auch Alternativen wie `Subethaedit` nutzen, was ich immer mache. Man muss aber auf jeden Fall `sudo` davor setzen, denn man kann nur mit `root`-Rechten die `sshd_config` verändern. Die `/etc/services` muss man zusätzlich ändern, wenn man den SSH-Port verändern will. Dazu muss sowohl die bei `/etc/services` als auch bei `/etc/sshd_config` der Eintrag für den SSH-Port geändert werden. Jedoch gibt es dabei immer ein paar Probleme. Auch ich hab diese. Angeblich soll es ausreichen bei der `/etc/services` die Zeilen

Code:

```
ssh          22/udp      # SSH Remote Login Protocol
ssh          22/tcp      # SSH Remote Login Protocol
```

zu ändern. Dies erreicht man durch den Befehl:

Code:

```
sudo pico /etc/services
```

Danach wird man nach dem Admin-Passwort gefragt. Wenn dies richtig eingegeben wurde, öffnet sich `pico`; ein Texteditor für die Konsole. Man scrollt dabei runter bis Port 22. Dort ändert man den Port von 22 auf einen anderen Port. Er sollte auf jeden Fall unter 65536 bleiben und nicht den Port eines aktiven Dienstes nutzen, der dann entweder blockiert oder zu komischen Nebenwirkungen führt. Ich nutze da einen **Portscanner**, der `Stealth FIN` und `ACK-Scans` durchführen kann. `Nmap` hat dann noch den kleinen Vorteil einen ping-losen Scan durchzuführen, womit die Firewall auch nichts blockt. Als Host geben wir logischerweise unseren Server ein, oder beim Server `localhost`, bzw. `127.0.0.1`.

Beim Scan-Typen `Stealth FIN` und der Ping-Art `no-ping` sollten wir auf alle offenen Ports stoßen. Offene Ports heißen immer aktive Service. Ist der Port geschlossen, ist der Dienst inaktiv. Somit nutzen wir den neuen SSH-Port, der dort nicht in der Liste steht. Aber wie gesagt... bei mir scheitert es aus unerfindlichen Gründen bei der Port-Änderungen.

Wie gesagt muss zusätzlich zur `/etc/services` die `/etc/sshd_config` für die Port-Änderungen abgeändert werden.

In der Zeile

Code:

```
Port 22
```

ändern wir die 22 in unseren neuen Ziel-Port um, den wir auch in `/etc/services` angegeben

haben.

Bei pico speichert man, indem man Ctrl+X drückt. Er fragt, ob die Änderungen übernommen werden sollen. Man tippt yes, bzw. ein y reicht, ein. Wenn wir gefragt werden als was es abgespeichert werden soll, und man übernimmt die Original-Datei und überschreibt man somit die Bestehende.

Alternativ kann man die Konfigurationsdatei auch anders bezeichnen und den SSH-Daemon, der auf dem Server läuft, per `sshd -f /ort/der/neuen/sshd_config` anweisen eine andere `sshd_config` für grundlegende Einstellungen zu nutzen.

Wenn es jemand hinbekommen haben sollte den SSH-Port manuell umstellen zu können, dann soll er es mir doch bitte verraten.

Da wir auch gerade noch in der `/etc/sshd_config` sind, ändern wir diese auch gleich noch weiter ab.

Falls die Zeile noch nicht existiert fügen wir die Zeile

Code:

```
PubKeyAuthentication yes
```

ein, bzw. ändern von no auf yes, falls es noch nicht geschehen ist. Damit wird die Authentifizierung per Schlüsseldatei erlaubt.

Wenn wir uns für die RSA-Verschlüsselung entschieden haben, setzen wir die Zeile

Code:

```
DSAAAuthentication no
```

ein, bzw. ändern von yes auf no.

Damit wird die Authentifizierung per DSA-Schlüssel verboten. Gleichzeitig erlauben wir aber mit der Zeile

Code:

```
RSAAAuthentication yes
```

die Authentifizierung per RSA-Schlüssel.

weiter nützliche Zeilen:

Code:

```
KeyRegenerationInterval 1800
```

Mit dieser Option weist man den SSH-Dienst an, alle 1800 Sekunden den Schlüssel zu erneuern. Wenn man anstelle einer beliebigen Zahl 0 schreibt, wird der SSH-Dienst angewiesen den Schlüssel niemals zu aktualisieren. Wenn die Option aktiviert ist, verhindert man, dass Hacker, denen ein Schlüssel bekannt ist, sich die komplette Verbindungszeit miteinhacken können. Sie werden durch die Regeneration des Schlüssels von der Verbindung getrennt und ausgesperrt.

Code:

```
ServerKeyBits 4096
```

Hiermit wird die Schlüsselgröße festgelegt. Es ist ähnlich wie bei der Erstellung des Schlüssels. Nur, dass hierbei mit der Option `KeyRegenerationInterval` ein Zusammenspiel stattfindet. Das ist also die Größe des neu generierten Schlüssels.

Code:

```
StrictModes yes
```

Es wird hiermit erreicht, dass der SSH-Dienst nähere sicherheitsrelevante Unterscheidungen durchführt. Dabei achtet er, bei sonst auch als standardmäßig aktivierter Form, auf gesetzte

Rechte auf Konfigurationsdateien und Ordner. Die Rechte- und Besitzer-Problematik ist dann essentiell wichtig und auch eine häufige Fehlerursache bei einer misslungenden Verbindung. Denn bei falsch gesetzten Rechten der *authorized_keys* oder des Login-Verzeichnisses verweigert der Dienst die Verbindung.

Code:

```
UsePrivilegeSeparation yes
```

Vor etwa 3 Jahren existierte in einer früheren OpenSSH-Version eine Sicherheitslücke, die durch *UsePrivilegeSeparation* verhindert wurde. Dabei konnte ein Eindringling direkt root-Rechte erlangen. Mit dieser Option kann man jedoch die Rechte-Verteilung vom SSH-Dienst genau aufteilen lassen. Es ist somit eine weitere Sicherheitsoption, die genutzt werden sollte, da sie mit *StrictMode* Hand in Hand geht

Code:

```
KeepAlive yes
```

Sollte eigentlich selbst erklärend sein... wenn es auf *no* steht, wird die Verbindung getrennt, sobald Inaktivität auftritt. Ansonsten wird die Verbindung getrennt, wenn das Terminal beendet oder die getunnelten Verbindungen getrennt wurden.

Code:

```
RhostsAuthentication no
```

Diese Option bildet eine andere Authentifizierung. Hierbei wird eine *rhost*-Datei angelegt und diese mit den Hosts, die sich auf unseren Servern verbunden haben, befüllt wird. Standardmässig ist sie deaktiviert und sollte es auch bleiben.

Es existiert noch die *RhostsRSAAuthentication*, die zusätzlich zur *Rhost*- eine Art RSA-Authentifizierung durchführt. Standard ist auch hier *no*.

Code:

```
ChallengeResponseAuthentication no
```

Diese Option würde Verbindungen als normaler User ohne Schlüssel zulassen bzw. beantworten, wenn aktiviert. Dies wäre anzuraten, wenn noch andere User auf den Server sich anmelden müssen. Hierdurch bekommen sie nämlich die Möglichkeit sich mit ihrem Benutzernamen und Passwort zu authentifizieren. Jedoch gab es da meines Wissens auch Probleme mit root-Login. Daher auch gleich noch der Befehl:

Code:

```
PermitRootLogin no
```

Hiermit verhindert man die direkte Anmeldung root-(Super)-User. Standardmässig ist dies auf *yes* gestellt, und sollte daher auf *no* gesetzt werden! Der unschlagbare Vorteil ist, dass man erstens Skript-Kiddies umgeht, die normalerweise alle Standard-User (und da ist root 100%ig drunter) durchgehen und Passwörter ausprobieren. Der Login *root* ist komplett blockiert durch diese Option und sollte auch wahr genommen werden. Der Hacker muss also beim Versuch unseren Server zu hacken den eigentlichen Login heraus bekommen, um dann festzustellen, dass zusätzlich mehr als nur ein Passwort, nämlich noch die Verschlüsselung knacken darf. Hat er dies geschafft, darf er erstmal knobeln wie das Passwort von *root* ist, um *superuser*-Rechte zu erlangen. Die vorgehensweise ist dann bei uns ähnlich. Man logt sich erst als Anwender mit einem Schlüssel an, und arbeitet dann entweder an entscheidenden Stellen per *su* oder bei kleinen Eingreifen mit *sudo*, und sonst als einfacher Anwender.

Code:

```
#PAMAuthenticatcionViaKbdInt no
```

Die genaue Arbeitsweise kann ich leider nicht erklären. Da sind die Quellen leider nicht so informativ... Es findet hier auf jeden Fall eine Interaktion mit der Option `ChallengeResponseAuthentication` statt. Hierbei wird das Passwort irgendwie per Tastatur-Interaktion aufgenommen und sich damit authentifiziert. Das lässt aber die Schlüssel-Authentifizierung null und nichtig werden. Standardmässig ist es auf `no` gestellt.

Code:

```
Protocol 2
```

Das sollte grundsätzlich(!) auf Protocol 2 beschränkt bleiben. Es ist jedoch auch auf Protokoll 1 erweiterbar. Die Protokolle sind mit Komma zu trennen.
Bsp.: `Protocol 1,2`

Code:

```
PermitEmptyPasswords no
```

Dies sollte auch grundsätzlich auf `no` gestellt sein/werden. Leere Passwörter sind das Einfallsloseste und Gefährlichste von allem. Ist diese Option auf `no` gesetzt, kann man sich nicht mehr einloggen als User, der kein gesetztes Passwort hat. Aber unter Unix-Systemen sollte dies auch auf normalen Wege gar nicht erst möglich sein.

Code:

```
PasswordAuthentication no
```

Hiermit deaktivieren wir die Passwort-Authentifizierung explizit. Aber wie schon bei anderen Optionen aufgefallen sein sollte, kann sich dies ins Gehege kommen mit zum Beispiel `ChallengeResponseAuthentication`. Auf solche Kleinigkeiten sollte man achten bei der Konfiguration, denn man kann ein inkonsistentis System riskieren.

Code:

```
LoginGraceTime 40
```

Hierbei wird die Zeit festgelegt in welcher sich der User authentifizieren und einloggen muss. Man sollte aber beachten, dass man die Zeit nicht zu niedrig wählt. Ein paar Datenpakete brauchen ihre Zeit von einem Punkt der Erde zum anderen. Dazu kommt dann die Zeit in der die Schlüssel verglichen werden. Zusätzlich wird auch die Passphrase-Eingabe dazu gezählt. Da sind schnell 30 Sekunden erreicht.

Code:

```
LogLevel VERBOSE
```

Hiermit legt man die "Tiefe" der mitgeschnittenen Aktivitäten, die in die Logs geschrieben werden, fest.

Es gibt folgende Möglichkeiten dabei: `QUIET`, `FATAL`, `ERROR`, `INFO`, `VERBOSE`, `DEBUG`. `INFO` ist der Standardwert, und `VERBOSE` sehr tiefgehend. `DEBUG` ist so stark mitschneidend, dass sogar private Aktivitäten der einzelnen User mitgeschnitten und protokolliert werden. Dies sollte man daher tunlichst lassen bei einem Firmen-Rechner oder dergleichen...

Code:

```
AllowUsers username
```

Hierbei werden die User eingetragen, mit denen man sich einloggen darf per SSH. Es gibt auch den Gegenpart *DenyUsers*, doch dieser steht hierarchisch unter *AllowUsers*. Wenn beide Befehle gleichzeitig genutzt werden, blockieren sie sich gegenseitig, und es kommt zu interessanten Nebenerscheinungen.

Code:

```
AllowGroups
```

Das gleiche in grün... man kann hier aber am besten mit der *Wildcard* arbeiten. Als *Wildcard* existieren `?` und `*`.

Beispiel: *AllowGroups Power?*

dies würde alle Mitglieder der Gruppen: PowerTeam, Power, PowerRangers etc. erlauben. Der Gegenpart zu dieser Option ist natürlich *DenyGroups*.

Sind alle Einstellungen getroffen, testen wir alles mit einem neuen Login per SSH auf unseren Server, nachdem alle Einstellungen übernommen wurden und der SSH-Daemon neugestartet wurde. Dies erreicht man indem man den Dienst bei Sharing de- und gleich darauf wieder -aktiviert.

Möglicher Geschwindigkeitsgewinn:

Wenn unser Test erfolgreich verlaufen ist, lohnt es sich die Einstellungen unseres Clients ebenfalls zu editieren. Dadurch erreicht man einen kleinen Geschwindigkeitsschub da die Einstellungen durch unseren Client schon vorgegeben sind und dieser nicht doch noch andere Möglichkeiten ausprobieren geht. Außerdem ist auch so unser Client während der SSH-Verbindung geschützt.

Es reicht, wenn wir per `sudo pico /private/etc/ssh_config` *DSAAuthentication* und *PasswordAuthentication* auf jeweils *no* setzen. Hier durch wird unser Dienst so eingeschränkt, dass wir uns nur noch mit RSA-Schlüsseln authentifizieren können. Wer SSH öfter braucht um sich auch auf andere Server mit nur einem Passwort oder DSA anzumelden, sollte seinen Dienst dann natürlich nicht einschränken.

3.3 Per SSH Anwendungen tunneln (Port-Forwarding)

Vielleicht kennt ein jeder das sichere HTTP-Protokoll, welches HTTPS genannt wird. Dies hat die selbe grundlegende Funktionsweise, die wir uns zu eigen machen wollen. Hierbei wird ein Schlüssel zwischen Server und Client ausgetauscht und nur über diesen Schlüssel kommuniziert man untereinander.

Wir haben nun schon die Schlüssel-Authentifizierung aktiviert und nutzen diese auch voll. Wenn wir nun Anwendungen von unserem Server auf unseren Client nutzen wollen, sollten wir auf eine sichere Verbindung achten. Die erreichen wir, indem wir die Übertragung der Daten per SSH *tunneln*. Tunneln ist hierbei wirklich sehr bildlich gesprochen.

Als kleines Beispiel:

ich nutze ~~MLDonkey~~ Webmin auf meinem Server. Dieses agiert im Hintergrund und über ein Webinterface eine bequeme Übersicht über Aktivitäten bieten. Mit Webmin kann man kurz gesagt den Server komplett fernwarten. Darunter fällt die User-, Festplatten- und Rechte-Verwaltung, um nur ein paar zu nennen. Es ist mit das mächtigste Tool um sein System auch in kürzester Zeit zu zerschneiden. Man kann das Webinterface einschränken. Entweder kann es nur vom localhost oder vom internen Netzwerk erreicht werden. Ferner kann es sogar für das Internet freigegeben werden, wenn der Router den Ziel-Port weiterleitet.

Das Webinterface von Webmin reagiert auf den Port 10000 standardmässig.

Wenn mein Server nun die IP-Adresse 192.168.1.2 hat, erreiche ist das Webinterface, wenn ich in den Browser: 192.168.1.2:10000 eingebe. Wenn ich es nur auf den localhost beschränke, funktioniert es logischerweise auch über: localhost:10000.

Ich möchte Webmin jetzt nur auf den localhost beschränken, um jegliches Sicherheitsrisiko zu minimieren.

Nun muss ich mir etwas einfallen lassen, wie ich trotzdem von meinem Client, der noch im selben Netzwerk steht, auf meinen Server zugreifen kann, um dann Webmin steuern zu können. Die Lösung steckt im Tunneln von Ports bzw. Diensten per SSH.

Hierbei verbinden wir uns per SSH von unserem Client auf unseren Server. Wir setzen als zusätzlichen Parameter dabei die Aufforderung, den Ziel-Port 10000 von unserem Server zu unseren Client weiterzuleiten.

Der Parameter würde dabei so aussehen (per RSA-Authentifizierung):

Code:

```
ssh -l username -L 10000:localhost:100 192.168.1.2
```

Mit diesem Befehl im Terminal wird der SSH-Dienst angewiesen sich auf die 192.168.1.2 zu verbinden mit unserem username. Weiter soll der Start-Port 10000 von unserem Server auf den localhost (unseres Client) auf den Ziel-Port 100 getunnelt werden.

Wenn die Authentifizierung erfolgreich war, öffnen wir bei unseren Client den Browser und tippen `localhost:100` ein. Jetzt öffnet sich das Webinterface von Webmin mit dem Login. Dies ist nun als sicherer anzusehen als zum Beispiel die Freigabe des Webinterfaces für das ganze Internet, denn nun wird unsere Verbindung mit dem Server über SSH verschlüsselt und geschützt. Durch die RSA-Schlüssel-Authentifizierung ist es auch nicht möglich, dass jemand Drittes unsere Verbindung mitlauscht oder gar mitsteuert. Diese Methodik kann auf alle Dienste angewendet werden, die die Möglichkeit bieten, Zugriffe per Netzwerk, oder gar vom Internet aus, zu steuern. Hierbei ist ein Webinterface immer am praktischsten, da dieses auch oft sehr sparsam gehalten ist, und somit auch keinen großen Netzwerkverkehr erzeugt.

Es können auch mehrere Ports bei nur einer SSH-Authentifizierung auf dem Server getunnelt werden.

Beispielsweise:

Code:

```
ssh -l username -L 4080:localhost:4080 10000:localhost:10000
```

Hierbei werden jetzt die Start-Ports 4080 und 10000 unseres Servers auf die Ziel-Ports 4080 und 10000 unseres Clients getunnelt.

Die Allgemeine Vorschrift für den Parameter -L ist:

Code:

```
ssh username@serverip -L Start-Port:ZielIP:Ziel-Port
```

Start-Port: Das ist der Port, der von dem aktiven Dienst auf dem Server genutzt wird. Wenn webmin auf dem Server läuft, dann läuft dieses standardmässig auf Port 10000. Wenn man den Port von Webmin auf 100 ändert, dann ist der Start-Port, der bei SSH gesetzt wird, auch 100.

ZielIP: Das ist das Ziel auf welches der Port von unserem Server hingetunnelt werden soll. Logischerweise schreibt man dabei eigentlich fast immer `localhost` oder `127.0.0.1`. Man kann jedoch auch eine andere ZielIP auswählen. Hierbei funktioniert jedoch anscheinend nur das interne Netz außerdem muss natürlich auch eine SSH-Verbindung zu der ZielIP bestehen.

Ziel-Port: Hier setzen wir den Port, auf den wir den getunnelten Dienst nutzen wollen. Man sollte hierbei am besten nicht Port 80 nehmen.... genauso wenig wie Port 8080, welcher oft von Proxys und anderen Internet-Diensten genutzt wird. Ansonsten gilt auch hier wieder: jeder Port ist frei nutzbar, solange er nicht von einem anderen Dienst belegt wird. Am besten nimmt man immer den Start-Port vom Server. Dann muss man sich auch keine krummen Zahlen merken.

iBook G4, 1.25Ghz, OSX 10.4, 80GB, 768MB

Debian PC-Server

iPod mini 2.G

Dell Widescreen 2005FPW

Geändert von Cyrics (Gestern um 22:29 Uhr).



Gestern, 13:17

3

Cyrics

Fuji



Registriert seit: 04.2005
Ort: Leipzig

3.4 SSH-Server schnell konfiguriert per SSH-Helfer

Hier noch einmal der Link zu der Donatware SSH Helper.

Mit diesem Tool lässt sich die `/etc/sshd_config` spielend per grafischer Oberfläche verändern und auch die Publickeys prima verwalten.

Wenn wir das Programm heruntergeladen und gestartet haben, möchte es von uns erstmal das Admin-Passwort. Ab hier sollte man überlegt handeln, genauso wie auch schon bei der manuellen Bearbeitung der Konfigurationsdateien. Mit sich widersprechenden Einstellungen, reisst man Löcher ins sonst eigentlich sichere System.

Alter: 21
Beiträge: 1.299



Wir brauchen erst einmal die Hilfe bei der Einrichtung des Servers.

Daher starten wir den SSH-Helfer am Server und gehen dabei wie folgt vor:

Am Anfang sollte sich der SSH-Helfer Wizard melden, der uns bei der Einrichtung des Client oder des Servers hilft. Ansonsten erreicht man diesen auch über das Menü *SSH->Setup Wizard*. Man folgt den Anweisungen und kann dabei noch SFTP, also sicheres Laden per FTP erlauben, und drückt dann rechts auf *Install Config*. Per *sudo* wird eine vorkonfigurierte *sshd_config* ins */etc* Verzeichnis geschoben. Wir gehen nun auf "Server Settings". Ist dies getan, können wir die config-Datei entweder manuell per Konsole wie oben beschrieben oder per SSH Helper editieren.



Beim SSH-Helfer kann man unter dem Register *Expert* eine manuelle Bearbeitung der *sshd_config* übersichtlich durchführen.

Unter *Settings* können wir dagegen grobe Sicherheitseinstellungen vornehmen.

Bei *Public Keys* befinden sich die drei angesprochenen Verschlüsselungsmethoden (RSA1, RSA, DSA) und die Authentifizierungsschlüssel, die man bei der Client-Konfiguration (Server Key) in der Wizard-Hilfe erstellen kann. Da ich mit den Server Keys aber wenig Erfahrung hab und ich eher den oben abgelaufenen Weg noch einmal gehen werde, werde ich hierzu keine Worte mehr verlieren.

Bei *Access Control* können wir die User- und Group-Verwaltung nutzen, die SSH ermöglicht den Zugriff auf dem Server weiter einzuschränken.

Um die Änderungen, die wir unter *Server Settings* vorgenommen haben, auch anzuwenden, müssen wir rechts oben auf *Install Server Setting* klicken, womit die zwischengespeicherte *sshd_config* ins */etc*-Verzeichnis kopiert und damit die Alte mit der Neuen überschrieben wird. Hierfür ist natürlich dann wieder das Admin-Passwort nötig um per *sudo* arbeiten zu können.

Um nun die selben Effekte, die in 2. beschrieben wurden per SSH-Helfer durchzuführen. Gehen wir zu unserem Client. Dort nutzen wir den Wizard (Menü *SSH -> Setup Wizard*) und folgen den Schritten für das *Setup Personal Secure Shell Identity*. Mit zwei Klicks weiter können wir die Key Parameter festlegen.



Hierbei greif ich wieder das Beispiel des RSA-Schlüssels auf. Wir wählen also aus dem Drop-Down Menü *SSH 2 - RSA* und darunter die Art unseres Schlüssels aus.

Hierbei wählen wir *Personal Key*. Hiermit wird dann wieder unser privater und unser öffentlicher Schlüssel-Paar erzeugt, wobei der private Teil auf unseren Client und der öffentliche Teil auf den Server wandert. Wir können auch noch einen Kommentar an die Schlüssel-Datei anfügen, wobei dieser Kommentar in Klarschrift an den Schlüssel angehängen wird. Der Kommentar ist sehr nützlich bei grafischen Programmen, wie auch dem SSH-Helper, da diese Programme die Schlüssel in der `authorized_keys` immer nach dem Kommentar alphabetisch anordnen werden. Der Standard-Kommentar ist sonst immer `username@ClientHost.local`.

Die zwei Passwort-Eingaben stellen die Passphrase-Eingabe dar. Hier kann man entweder die Felder leer lassen und somit die Datei unverschlüsselt lassen oder eben ein zusätzliches Passwort zur Authentifizierung am Server schaffen.

Wir generieren dann unseren neuen Schlüssel, der ohne Nachfrage in unser `~/.ssh/`-Verzeichnis kopiert wird. Wenn hier also Schlüssel mit der Standard-Bezeichnung drin liegen, werden die rigoros überschrieben... leider.

Wenn wir nun den Wizard beenden und auf unser "*Username's Settings*" gehen und dort auf *My Keys*. Können wir uns von allen 3 möglichen Verschlüsselungen den public-Key anzeigen lassen. Wenn wir hier *SSH 2 RSA* aus dem Drop Down Menü auswählen finden wir unseren Schlüssel wieder. Den können wir nun, wenn wir am Server sind, zu unsere lokalen *Authorized_Keys* hinzufügen lassen, einen neuen Schlüssel generieren lassen oder den bestehenden Löschen.

Wenn den Schlüssel zu unseren lokalen *Authorized_keys* hinzufügen lassen, finden wir den Schlüssel im Register *Authorized Keys* unter *Protocol 2 Keys* wieder. Hier finden wir den Schlüssel nach dem optional hinzugefügtem Kommentar. Per "+" und "-" lassen sich weitere Schlüssel hinzufügen oder entfernen.

Bei *My Keys* müssen wir sonst den public-Key exportieren, um ihn zum Server zu übermitteln. Hierbei speichern wir ihn entweder auf einer gemounteten Freigabe auf dem Server oder speichern ihn lokal erst einmal zwischen um ihn wieder per Terminal und scp-Befehl zu verschicken wie bei 3.2.3 erklärt.

4. DynamicDNS

4.1 Allgemein Dynamic DNS und dessen Vorteile

DynamicDNS bedeutet einen weltweit verfügbaren eindeutigen Eintrag in einem Register um eine wechselnde Ausgangs-IP des Routers, und damit den Router selbst, zu jeder Zeit ohne Wissen der gerade aktuellen IP erreichen zu können. Wikipedia kann das Ganze sicher verständlicher erklären. Falls sich viele darunter noch nichts vorstellen können, sei aber hier noch ein Beispiel vorgebracht. Ich habe meinen Server in meinen Augen ausreichend sicher gestaltet und die nötigsten Dienste für das Internet freigegeben. Die Anfragen an diese Dienste werden an bestimmte Ports an meinem Server gerichtet. Diese Anfragen kommen nur durch, wenn der Router dementsprechend konfiguriert ist (4.2 und 5). Ich will mir nun nicht tagtäglich die derzeitige AusgangsIP meines Routers, und damit meines Servers erfragen. Um dies zu umgehen richte ich mir bei dyndns.com einen Account ein mit dem ich

einen Domain-Namen frei mit kleinen Einschränkungen auswählen kann. Dieser Account verknüpft nun die aktuelle AusgangsIP meines Routers mit meiner Wunsch-Domain. Wenn ich nun also den Domain-Namen eingebe, wird bei dyndns.com nachgeschaut mit welcher Ziel-IP-Nummer dieser verknüpft ist. Es wird nun die aktuelle IP-Nummer anstelle des Domain-Namens verwendet, aber dies alles ohne die Arbeit dahinter mitzubekommen. Dies ist ein wunderbarer Service, den ich euch hier ein bisschen näher erläutern und bei der Konfiguration behilflich sein möchte.

Mit dem einfach zu merkenden Namen kann man sehr leicht den Server erreichen ohne ständig sich die schwierig einzuprägende IP merken zu müssen. Und hier sind wir schon beim ersten großen Vorteil! Ein Name ist leichter ins Gedächtnis aufzunehmen als eine mehrstellige Nummer. Angeblich kann der durchschnittliche Mensch sich nur 6 aufeinander folgende Zahlen merken, die er nur für einen kurzen Zeitraum wissen muss.

Bei unseren Internet-Providern bekommen wir alle 24 Stunden die neue IP-Nummer zugeordnet und müssen diese heraus suchen und wieder neu einprägen.

Vorteil 2: der Domain-Name bleibt immer der Gleiche!

Es ist dazu umständlich und setzt jeden Tag ein Stück Arbeit voraus. Da wir alle samt gerne faul sind, lassen wir das automatisch erledigen entweder vom Router oder dem Server.

Vorteil 3: einmalige Einrichtung und Konfiguration und kein weiterer Eingriff wird nötig sein.

Das sollen genug Vorteile sein. Ein Nachteil entsteht indem Sinne nicht. Manche befürchten durch den DynamicDNS-Eintrag öfter von Crackern und deren Skripten besucht zu werden, aber es ist dabei egal, ob man bei Dynamic DNS drin steht oder nicht, denn hier laufen Zufallskripte, die einen vordefinierten Adress-Bereich abgrasen lassen.

4.2 Dynamic DNS - Account einrichten

Wie oben schon genannt, kann man sich unter dyndns.com einen Account anlegen. Ich will nun keine Werbung für diese Seite machen. Es gibt sicher noch einige andere Möglichkeiten. Ich will einzig und allein die Sache ansich erstmal einrichten. [Hier](#) kann man sich nämlich den Account erstellen. Dabei wird ein Benutzername, Email-Adresse und Passwort benötigt. Alle drei Dinge brauchen wir dann auch später noch.

Wenn unser Account eingerichtet ist, loggen wir uns ein. Wenn man eingeloggt ist, kann man auf [My Services](#) gehen. Dort können wir unseren kostenlosen Account "upgraden" zu einem kostenpflichtigen Account mit dem man mehrere unbegrenzte DynamicDNS und noch viele andere Services nutzen kann. Auf der linken Seite finden wir im Menü den Link [Add Host Services](#). Hier klicken wir nun rechts in der Liste auf [Add Dynamic DNS Host](#).

New Dynamic DNSSM Host

Hostname:	<input type="text"/>	<input type="text" value="homeftp.net"/>
IP Address:	<input type="text" value="89.57.46.35"/>	
Enable Wildcard:	<input type="checkbox"/>	
Mail Exchanger (optional):	<input type="text"/>	<input type="checkbox"/> Backup MX?

Hier wird nun ein Formular geöffnet indem wir unsere Wunsch-Domain bei Hostname auswählen. Dazu wird eben ein Subdomain-Name von uns ausgewählt mit der eigentlich Domain und deren Endung am Ende.

Als Beispiel nehmen wir *osx-server* als Hostname mit der Endung *homeftp.net*.

Als *IP Address* wird unsere derzeitige AusgangsIP-Nummer angezeigt. Diese bestätigen wir bzw. nehmen wir dann einfach hin, sollten aber bedenken, dass die Domain mit der IP-Adresse unseres jetzigen Routers verbunden ist. Steht der Rechner im selben Netzwerk wie unser Server, bzw. ist am selben Router angeschlossen, ist dies kein Problem.

Mit *Enable Wildcard* ist gemeint, dass jeglicher Aufruf auf unsere Domain an unsere IP weitergeleitet wird. Ansonsten würde nur der Aufruf *osx-server.homeftp.net* auf unsere Startseite des Servers weitergeleitet. Dagegen würde der Aufruf *irgendwas.osx-server.homeftp.net* keine Weiterleitung hervorrufen. Nutzt man den Server nur als SSH-Server ist dies natürlich kein Problem. Wenn man aber auf Sub-Sub-Domains oder dergleichen angewiesen ist, sollte man die Option *Wildcard* aktivieren. Wer sonst noch einen Mail Exchanger betreibt, sollte dessen Hostnamen in das entsprechende Feld eintragen. Aber ich glaub das wird hier kaum jemanden betreffen, oder irre ich mich?

Ist dies eingerichtet, speichern wir unseren Dynamic DNS Host. Wir wollen nun aber auch noch einen Schritt weiter und wollen, dass ein Dienst unsere IP-Nummer täglich aktualisiert, wenn sie sich

geändert hat.

4.3 Router für DynDNS konfigurieren

Es kommt auf das jeweilige Router-Modell an, aber heutzutage sollte so ziemlich jeder Router die automatische Aktualisierung eines DynDNS-Accounts beherrschen. Selbst mein T-Sinus 111 kann dies.... und der ist der letzte Schrott.

Meistens findet man DynamicDNS oder DDNS oder DynDNS unter *Netzwerkeinstellungen*. Hier sollte sich einer der drei genannten Begriffe auffinden lassen. Dort kann man dann meistens nur zwischen ein paar wenigen Anbietern von DynamicDNS-Accounts auswählen. Fast immer ist DynDNS.org in der Liste zur Auswahl.

Als erstes muss man beim Router diesen Dienst aktivieren. Dann wählt man den DynamicDNS-Anbieter aus einer Liste, die eigentlich bei jedem Router existiert, oder gibt explizit dyndns.org ein, welches sonst ihre Ursprungs-Adresse ist.

Weiterhin müssen nun der Domain-, bzw. Hostname, sowie Username und Passwort eingegeben werden. Wenn nur ein Username und Passwort verlangt wird, weiß ja der Router nicht, welchen Hostname er mit der aktuellen AusgangsIP-Nummer aktualisieren soll. Daher ist dieser zwingend notwendig, wie natürlich auch die anderen Angaben. Manchmal wird auch anstelle des Username die Email-Adresse, mit der man sich bei dyndns.com registriert hat, vom Router gefordert.

Die Router mit neuer gut-gepflegter Firmware bieten nun noch weitere Optionen, wie oft die IP-Adresse aktualisiert werden soll etc. Manchmal wird auch hier nochmal gefragt, ob Wildcard aktiviert werden soll und dergleichen. Das hängt aber nun von den Vorlieben jedes einzelnen Benutzers, aber auch von den Möglichkeiten des Routers ab.

4.4 Server mit DynDNS-Dienst einrichten

Wenn der Router keine solche Möglichkeit bietet den DynDNS-Account regelmässig automatisch aktualisieren zu lassen. So müssen wir den Server mit einem solchen Dienst einrichten. Hierbei sollte nicht unerwähnt bleiben, dass wenn zwei Dienste, d.h. sowohl auf dem Router als auf dem Server, laufen, die den DynDNS-Account aktualisieren sollen, dann löschen sich diese Dienste gegenseitig aus. Es ist verrückt, aber es funktioniert einfach nicht. Daher sollte man nur auf einen Dienst zurückgreifen und keinesfalls auf zwei bauen.

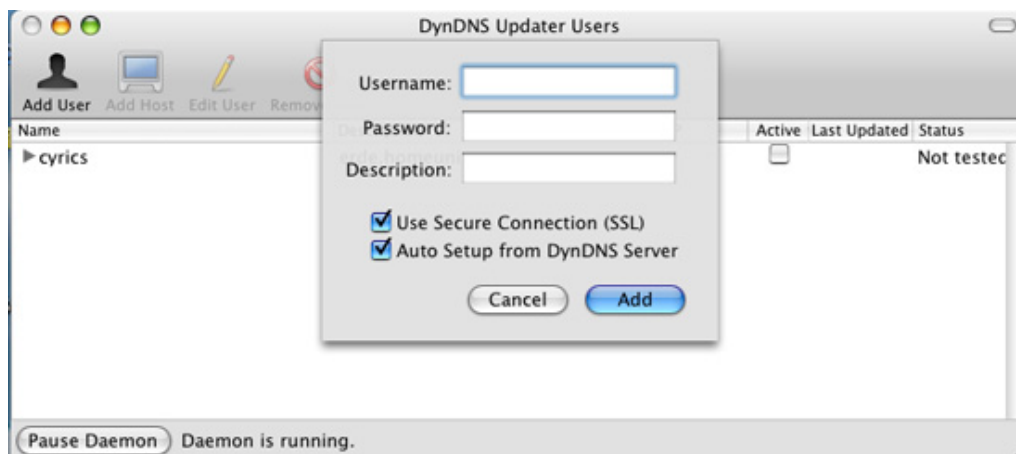
Da dies nun gesagt wurde, wäre mein Tipp eines DynamicDNS-Dienstes entweder mein bevorzugter **DynDNS Updater 1.2** oder der fast gleiche **DNSUpdate 2.7**.

Beide nutzen den selben Daemon. Bitte nicht von diesem Wort abschrecken lassen. Damit ist einfach nur der Dienst gemeint, der im Hintergrund, ohne das es der Nutzer mitbekommt, läuft. Beide Programme lassen sich bei Nicht-Gefallen sehr leicht wieder im Menü deinstallieren.

DNSUpdate hat den Vorteil weitere Anbieter von Dynamic DNS aufzuführen in der Liste. *DynDNS Updater* beschränkt sich hierbei komplett nur auf DynDNS.com.

Jedoch finde ich persönlich den *DynDNS Updater* besser, da dieser auch gleich ein Widget mitliefert und von der Bedienung besser ausgearbeitet ist. Außerdem funktioniert hier die verschlüsselte Verbindung zum DynDNS-Account auch, und wird nicht wie bei DNSUpdate nur grau unterlegt dargestellt.

Zu den Problemen beider Programme: Wenn sich der angemeldete Benutzer abmeldet, stürzt der Daemon ab. Somit wird während dieser Zeit der DynDNS-Account nicht aktualisiert. Der Absturz des Daemons kann nur behoben werden durch einen Neustart bei dem der Daemon neu initialisiert wird. Da ich meinen Benutzer nie abmelde, stört mich das eigentlich auch nicht. Für diejenigen, die dies öfter tun, sei das hier eben gesagt. Einen Server meldet man sonst auch nicht ab! Da meldet sich nicht mal ein User an! Nur zur Erinnerung... der Daemon kann im Menü unter Preferences eingestellt werden, wann dieser gestartet werden soll. Dies sollte für den Server bereits während des Hochfahrens geschehen. Daher stellen wir dies auch hier ein.



Mit *Add User* wird hier ein Benutzer hinzugefügt für DynDNS.org. Bei DynDNS Updater ist dies spielend leicht, da man hier die Daten von DynDNS.org synchronisieren lassen kann und so alle eingerichteten Hostnames abgleichen kann. Wenn die Verbindung erfolgreich hergestellt wurde, erscheint unser Hostname bei DynDNS.org in der Liste unter unseren Benutzernamen. Mit einem Häkchen bei Active können den DynamicDNS-Eintrag aktivieren und so wird dieser dann mit unserer AusgangsIP-Nummer nun automatisch aktualisiert. Zusätzlich kann man sonst bei anderen Dynamic DNS Einträgen sehen wie der Status dieser Domains ist, ob diese noch genutzt werden oder schon als inaktiv gelten etc.

Mit dem Widget lässt der aktuelle Status des Daemons und des Dynamic DNS Eintrages beobachten. Also ansich alles sehr einfach, übersichtlich und doch sehr praktikabel gestaltet.

5. Die nötige Router-Konfigurierung

Diese Router-Konfigurierung ist nötig, sobald man seinen Server auch von außerhalb, das heisst von irgendeinem Punkt der Erde über das Internet auf seinen Server zugreifen möchte, wie ich es gerne tu. Dabei muss die Firewall des Routers neu konfiguriert werden wie auch eine Paket-Weiterleitung für Anfragen für bestimmte Ports (und somit Dienste) an unseren Server weitergeleitet und beantwortet werden.

Die Problematik ist ja folgende: Unser Server hat eine IP-Adresse nach außen. Besucht man nun eine Webseite muss der Paketfilter des Routers registrieren, dass wir eine Anfrage an meinetwegen apfeltalk.de gestellt haben und schickt die erhaltene Antwort von apfeltalk.de dann an unseren Router und dieser an unseren Mac. Wenn nun 10 Rechner verbunden sind mit dem Router muss der von 10 Rechnern die Pakete verteilen. Soweit ist dies alles kein Problem. Jedoch was passiert mit Anfragen, die wir selbst nicht gestellt haben, sondern von außen nun von jemand anderen an unseren Mac im Netzwerk über den Router geschickt werden? Dieser weiß nun prinzipiell überhaupt nicht welchem Mac er das Paket zuordnen soll, und hat daher im Paket-Filter und in der Firewall drin stehen, dass unaufgeforderte Pakete geschluckt werden.

Als Beispiel sei hier wieder mein laufender Dienst Webmin aufgegriffen. Dieser läuft auf dem Port 10000. Mein Server als auch mein Router sind im Ruhezustand. Nun läuft auf meinem Router der DynDNS-Dienst und so greife ich auf meine Domain os-x.homeftp.net zu. Der Router erkennt diese Anfrage und blockiert sie sofort.

Dies liegt erst einmal an der Firewall.

Wir müssen also erst einmal die Firewall neu konfigurieren, dass sie Anfragen auf bestimmte Ports nicht mehr prinzipiell schlucken soll! Bei manchen Routern reicht es die NAT (**Network Address Translation**) umzukonfigurieren und die Firewall wird dadurch automatisch angepasst. Manche müssen jedoch per Hand manuell eingestellt werden. Dies findet man oft unter dem Punkt *Sicherheit* oder *Netzwerkeinstellungen*. Hier muss eine neue Regel erzeugt werden, welche Anfragen auf unseren SSH-Port erlaubt. Also die grundlegende Einstellung der Firewall bleibt erhalten, dass alle Pakete, die unaufgefordert gesendet werden, geschluckt werden, wobei eben dann eine Ausnahme von uns erstellt wird, die ein Durchlassen von Paketen auf einen bestimmten Port erlaubt.

Nun müssen wir nach der Option NAT bei Netzwerkeinstellungen o.ä. suchen. Hier müssen wir auch eine neue Regel erzeugen. Wenn NAT nicht verfügbar ist, existiert vielleicht sonst die radikale Möglichkeit, die DMZ auf den Server zu übertragen. Hierbei wird der Server sozusagen aus dem internen Netz heraus genommen und steht dem Internet voll zur Verfügung. Im Gegenzug steht er auch mit allen Angriffen voll im Beschuss, da hier der schützende Faktor Router wegfällt. Dieser leitet bei der DMZ prinzipiell alle Anfragen erst einmal komplett an den Server weiter, wenn er in der DMZ steht. Wenn man nur diesen Ausweg kennt, sollte man entweder überlegen sich einen neuen Router zu kaufen, oder seinen Server wirklich sicher zu gestalten und möglichst wenig Dienste auf diesen zu laufen. Besonders keinen HTTP-Server mit Plugins wie PHP, MySQL etc.

Wenn wir NAT finden, müssen wir vielleicht noch irgendeinen Unterpunkt auswählen (meistens Virtueller Server oder Adressumsetzung). Wir müssen die Funktion, bei der wir eine Anfrage an einen Port über eine von uns bestimmte IP-Nummer im Netzwerk mit TCP- oder UDP-Paketen stellen können.

TCP-Pakete sind sozusagen antwortende Pakete um es mal schnell und leicht zu erklären. UDP sind das krasse Gegenteil gegenüber TCP. UDP sind nicht-antwortende Pakete. Sie werden verschickt und verlieren sich im Raum. Bei TCP wird das Paket geschickt und wenn es das Ziel erreicht, wird eine Antwort an den Absender zurückgeschickt, damit dieser bescheid weiss (natürlich nur das System), falls die Übermittlung fehlerhaft war, wird der Vorgang wiederholt. Bei UDP weiss man nicht, ob das Paket überhaupt jemals ankam.

SSH ist daher, wie man sicher leicht erraten kann, ein Dienst, der TCP als "Versende"-Methode nutzt. Wenn wir die betreffende Option beim Router gefunden haben, tragen wir die Ziel-Adresse unseres Servers im Netzwerk ein. Dies war im aller ersten Kapitel genomme Beispiel-IP-Nummer 192.168.1.2. Wenn der Router die Möglichkeit bietet zwischen öffentlichen und privaten Ports, oder internen und äußeren Ports, zu unterscheiden, dann sollten wir davon Gebrauch machen. Hierbei wird der öffentliche Port derjenige sein auf dem der Router von außen angesprochen wird. Dieser leitet die Anfrage auf diesen öffentlichen Port dann weiter an die IP und an den privaten Port des Servers.

Als Beispiel:

IP des Servers: 192.168.1.2
 öffentlicher Port: 6666
 privater Port: SSH auf Port 22

Ich bin nun in einem anderen Netzwerk als mein Server und greife auf meine DynDNS-Domain zu. Damit greifen wir auf die IP-Nummer des Routers zu. Meine Anfrage enthält die Bitte:

Code:

```
ssh -l username -p 6666 -L 10000:localhost:10000 osx-server.com
```

. Der Router erkennt nun die Anfrage auf Port 6666 und weiss durch seinen Eintrag im NAT, dass er Anfragen auf Port 6666 auf Port 22 von unseren Server 192.168.1.2 weiterleiten soll. Dies wird natürlich getan und der Server antwortet und der Router leitet die Anfrage an uns zurück. Nun läuft der Netzwerk-Verkehr und der Router leitet die Ports dabei ständig hin und her.

Dies ist natürlich auch alles kein Schutz gegen Hacker-Angriffe, da diese schnell den SSH-Port erkennen können anhand der Antwort, die von diesem Port kommt, aber wenn der Port extra hoch ausgewählt wird, haben viele Port-Scanner schon Probleme zu differenzieren, bzw. viele Portscanner scannen auch nur den unteren Port-Bereich gründlich, da hier die wichtigsten Dienste laufen.

Diese Einstellungen reichen. Alle Anwendungen die wir nun sonst noch von außen her erreichen wollen würden, werden wir dann per SSH tunneln. Soweit es sich um eine Anwendung handelt, die darüber zu tunneln geht (was eigentlich bei fast allem funktioniert).

iBook G4, 1.25Ghz, OSX 10.4, 80GB, 768MB

Debian PC-Server

iPod mini 2.G

Dell Widescreen 2005FPW

Geändert von Cyrics (Heute um 01:57 Uhr).



Gestern, 13:17

#4

Cyrics

Fuji



Registriert seit: 04.2005

Ort: Leipzig

Alter: 21

Beiträge: 1.299



6. VNC

6.1 Vergleich SSH und VNC

Wer nun noch nicht müde ist etwas mehr zu lesen (obwohl wir hier schon an die 60.000 Zeichen heran kommen) über Remote-Lösungen, stell ich hier auch noch VNC vor.

Viele können sich einfach nicht mit der Konsole anfreunden und bestehen darauf auf einer grafischen Oberfläche zu arbeiten. VNC (**Virtual Network Computing**) bietet dies. AT&T hat diesen Dienst in damaliger Zeit entwickelt um schnell und einfach die Oberfläche eines hilflosen Benutzers zu steuern.

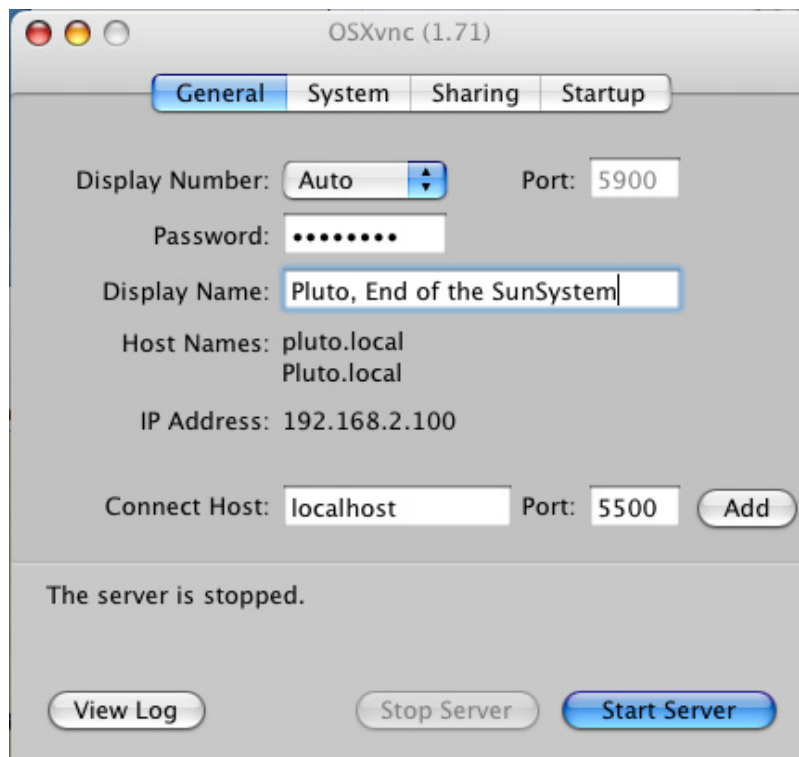
Dies alles hat seinen Preis, nämlich wesentlich mehr Netzwerk-Verkehr. Man übermittelt jegliche grafische Darstellung über diese lächerlich kleine Leitung. VNC läuft über eine 100Mbit/s-Leitung sehr bescheiden. Es lässt sich damit arbeiten, aber Spass macht es nicht wirklich. Außerdem sind in jedem System verschiedene Versionen von VNC implementiert, die manchmal zu Übersetzungsproblemen zwischen den Versionen führen. Dies bringt weiteren Geschwindigkeitsverlust und Hänger.

Man kann ein Passwort für die Verbindung von VNC erzeugen, jedoch ist dieses nicht wirklich sicher. Aus diesem Grund wird auch VNC eher immer im Zusammenhang mit SSH genutzt. VNC lässt sich hierfür wunderbar verwenden. VNC-Server läuft standardmässig eigentlich immer auf Port 5900 und der VNC-Viewer lauscht immer auf Port 5500 doch dazu mehr bei 6.4.

6.2 VNC - Server

VNC - Server für OSX kann man [hier](#) u.a. herunterladen. Dieser ist für unsere Zwecke eigentlich ausreichend.

Wenn wir diesen heruntergeladen und initialisiert haben, stoppen wir am besten erst einmal den Server, falls er sich automatisch gestartet hat. Außerdem sollten wir unter *Systemeinstellungen* -> *Sharing* -> *Dienste* -> *Apple Remote Desktop* aktivieren damit VNC überhaupt durch die Firewall gelassen wird.



Wir sehen bei *General* Display Number, Port, Password, Display Name, Host Names, IP Address, Connect Host and Port again.

In dieser Reihenfolge:

Display Number: hier können wir entweder den aktuellen Bildschirm freigeben, oder einen von uns bestimmten. Nun wird vielleicht nicht jeder diese Besonderheit der Unix-Grundsysteme kennen. Unter jedem Unix-System existieren unendlich viele Arbeitsplätze, bzw. Desktops. Man kann bei OSX auf maximal 10 Arbeitsplätzen gleichzeitig arbeiten und zwischen diesen Desktops hin- und her wechseln. Dafür ist zum Beispiel **Desktop Manager** (Beta-Version) sehr zu empfehlen.

Port: Dies ist logischerweise der Port auf den der Dienst dann laufen wird. Diesen kann man auch verstellen wenn man will. Jedoch sollte man hier wieder auf laufende Dienste im Hintergrund achten, die möglicherweise den selben Port dann nutzen könnten.

Password: hier kann man ein Passwort definieren, welches der Gegenüber wissen muss um an der Sitzung teilzunehmen. Doch dieses ist relativ schnell geknackt beim Mitlauschen an einer Sitzung und dem dabei entstehenden Datenverkehr.

Display Name: unser Client kriegt als Ziel diesen Namen angezeigt und sollte auch so eindeutig sein, dass der Client sich nicht wundert und auch sicher sein kann auf dem richtigen Server zu sein.

Host Names: der VNC-Server lässt sich entweder über diese Host Names erreichen oder über die IP-Adresse. Aber dies funktioniert natürlich auch nur im internen Netzwerk.

IP-Adress: die aktuelle IP-Adresse, die unser Server im internen Netz führt.

Connect Host und Port: hier kann eine Einladung ausgesprochen werden zu einem Host und dessen Ziel-Port unserer Wahl. 5500 ist der Standard-Port auf dem der VNC-Viewer lauscht. Wenn eine Einladung ausgesprochen wurde, erhält der VNC-Viewer die Nachricht, dass er eingeladen wurde und mit einem Klick an der Sitzung teilnehmen kann.

Unter dem Register *System* finden wir grundlegende Einstellungen, die entweder unterdrückt oder hervorgerufen werden sollen während der Verbindung.

Bei Video können wir entweder Sleep-Mode, das Dimmen des Bildschirms oder den Bildschirmschoner erlauben.

Weiterhin kann man Maus-Taste 2 und 3 tauschen lassen. Jedoch erschliesst sich mir hier nicht der Sinn dieser Option.

Unter Protocol wird die Version und damit auch die Art der Qualität bestimmt. Neuer heisst nicht immer besser.

Unter dem Register *Sharing* können wir Einstellungen treffen, die zwischen Client und Server

ausgetauscht werden.

Besonders interessant ist hierbei bei Display Sharing, wo wir den Client entscheiden lassen können oder das Desktop niemals transferieren zwischen Client und Server. Außerdem können wir mehrere Clients gleichzeitig zu uns verbinden lassen mit einem Häkchen bei *Keep existing client[...]*.

Wir können dann weiterhin Maus und Tastatur des Clients verbieten und so hat der Client nur die Möglichkeit bei den ausgeführten Aktionen auf dem Desktop zuzugucken.

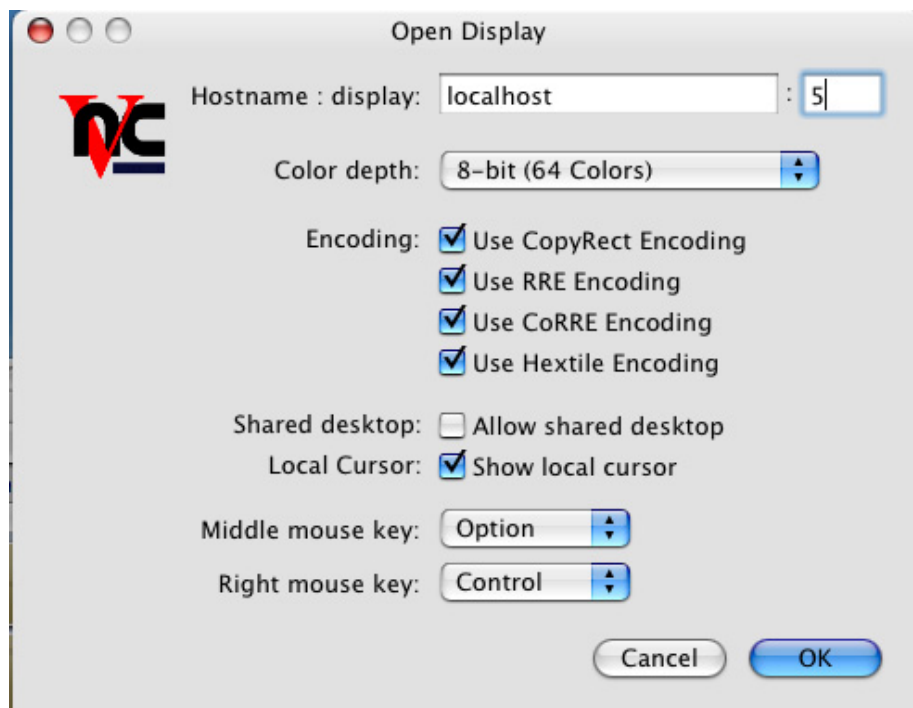
Wir können das Anbieten des VNC-Dienstes auf das interne Netzwerk beschränken und diesen Dienst per Bonjour im Netzwerk ankündigen.

Bei *Startup* können wir einstellen unter welchen Bedingungen VNC-Server den Dienst starten soll. Mit einem Klick auf *Configure Startup Item* wird VNC-Server in die Startup Items eingetragen und somit beim Hochfahren bereits gestartet. Das hat den Vorteil VNC auch nutzen zu können, wenn noch kein Benutzer angemeldet ist.

Außerdem können wir das automatische Starten von VNC beim Start von VNC-Server hier unterbinden. Wenn wir den Benutzer per Fast User Switch wechseln, bleibt unser eigentlicher Account ja trotzdem angemeldet. Hier kann man nun erreichen, dass die VNC-Verbindung beendet wird, wenn der Benutzer gewechselt wird. Außerdem kann durch eine Option der Server sofort wieder gestartet werden, wenn er durch einen Fehler unerwartet beendet wurde.

6.3 VNC - Client

Den VNC-Viewer oder Client finden wir [hier](#). Dieser ist sehr schnell konfiguriert einsetzbar. Wenn dieser heruntergeladen und initialisiert wurde, müssen wir erst einmal ins Menü auf *Display* -> *Open* gehen.



In diesem Dialog können wir den Ziel-Host eingeben auf dem der VNC-Server läuft sowie dessen Ziel-Port.

Bei Encoding können wir alle Encoding-Varianten aktivieren, die wir auch nutzen wollen. Es kommt hierbei auf die VNC-Server-Version an wie diese die Grafik-Pakete packt und verschickt. Es empfiehlt sich einfach alle 4 Dienste aktiviert zu lassen. Wenn man VNC per Konsole nutzt, sind noch weitere Encoding-Typen möglich.

Shared Desktop sollten wir prinzipiell deaktivieren. Hierbei wird das Hintergrundbild des Servers und andere Spielereien nicht übermittelt. Dies erspart natürlich einen ziemlich großen Teil des Datenstromes, der sonst auftritt.

Weiterhin können wir unseren lokalen Cursor deaktivieren lassen, solange wir mit dem Server verbunden sind. Außerdem lassen sich die mittlere und rechte Maustaste von uns individuell definieren mit Tastatur-Tasten.

Wenn die richtige IP-Adresse und Port-Nummer eingegeben wurde, und der Server ein Passwort verlangt, dann wird dies erst verlangt, wenn die Verbindung schon halb aufgebaut ist. Hier muss man eben das Passwort dann eintippen.

Und nun lässt sich der entfernte Rechner bequem per VNC steuern.

6.4 VNC - Optimierung

Wenn wir per SSH arbeiten, dann sollten wir per SSH die Verbindung an unseren Server schicken mit der Aufforderung den Port 5900 im günstigsten Fall auf einen anderen Port weiterzuleiten. Beispiel:

Code:

```
ssh -l username osx-server.homeftp.net -L 5900:localhost:6000
```

Jetzt wird der Port 5900 von unserem Server auf unseren lokalen Port 6000 getunnelt. Jetzt bräuchten wir nur den VNC Viewer öffnen und eine Anfrage an die IP-Adresse *localhost* mit dem Zielport 6000 zu schicken. So ist die Verbindung zwischen Client und Server schon einmal gesichert über SSH. Dies alles setzt aber natürlich voraus, dass der VNC-Server auf unserem Server bereits läuft und auch antworten kann. Außerdem ist es auch hier angebracht trotzdem ein Passwort für die Sitzung zu setzen. Lieber immer mehr als nur einen Schutz nutzen!

Wenn wir die Grafik-Anforderungen noch stark herunterschrauben wollen, schaffen wir dies indem wir auf 8-Bit-Grafikdarstellung (64 Farben) herunter gehen. Dann haben wir auch schon die Minimal-Anforderungen erreicht. Wenn unser Server nur 16kbit/s Upload geniessen kann, ist es trotz dieser Einstellungen nicht unbedingt angenehm mit VNC zu arbeiten.

7. Alternativen: KVM-over IP

Eine besondere Alternative auf die mich pete gebracht hatte ist ein *KVM-Switch over IP* oder auch *KVM over on the Net* genannt.

Vielleicht kennt ein jeder den klassischen *KVM-Switch*. Mit Hilfe dieses KVM-Switches braucht man nur ein Eingabegerät und kann mittels Knopfdruck (modellabhängig) zwischen bis zu 8 PCs hin- und her-wechseln. So braucht man auch nur einen Monitor, eine Tastatur und eine Maus und kann damit 8 PCs gleichzeitig bedienen. Dies im normalen Büro-Alltag ein wahrer Segen, da dort oft ein PC nur für eine spezielle Aufgabe genutzt wird und während der übrigen Zeit dieser einfach nur rumsteht oder seine Aufgaben löst. Während dessen wechselt man zu anderen PCs um dort die Arbeiten voranzutreiben.

Nun wird diese Lösung hauptsächlich für Administratoren im Server-Raum genutzt. Hier stehen einige laute Maschinen und immer nur ein Eingabe-Gerät. Da viele Administratoren aber nicht wirklich Spass haben in dieser lauten heißen Hölle voll mit Servern existiert die Möglichkeit eines KVM-Switches over IP!

Hierbei kann entweder der alte KVM-Switch erweitert werden oder mit einer solchen Komplet-KVM over IP-Lösung ausgetauscht werden.

Die Funktionsweise ist nun folgende:

Der KVM over IP (ich schreib jetzt nur noch KVMIP)-Switch kann per VNC-Viewer oder per java-fähigen Browser angesprochen werden. Der Switch hat dabei eine eigene Netzwerk-IP. Am besten ist es per Browser, da sich hier ein benutzerfreundliches Menü ansprechen lässt, welches per SSL verschlüsselt wird. Über dieses Menü müssen wir uns identifizieren und dann die angeschlossenen Rechner, bzw. IP-Adressen auswählen. Der KVMIP-Switch leitet diese Anfrage nun weiter an diese Ziel-IP-Adresse. Der absolute Vorteil dessen ist nun, dass man sogar auf den Rechner zugreifen kann, obwohl dieser nicht mal richtig hochgefahren ist. Ich kenne hierbei nun den Switch von Aton, welcher die Netzwerk-Erweiterung des bereits bestehenden KVM-Switches bildet. Es wird im selben Netzwerk die Anfrage an den KVMIP-Switch gestellt, dieser leitet diese Anfrage an den KVM-Switch dahinter weiter und gibt ihm die Anweisung auf Schalter 1 meinetwegen zu gehen.

Mit gut ausgestatteten KVMIP-Switches lassen sich BIOS steuern oder sogar die Rechner einschalten. Ich finde dies persönlich eine Revolution der Remote-Lösungen. Man kann als Administrator von jedem Punkt der Erde seine Netzwerke pflegen und hegen. Nur bei einem physischen Fehler/Schaden muss man höchst persönlich eingreifen oder weist eine Hilfskraft, die in der Nähe arbeitet an.

Diese KVMIP-Switches haben trotz allem das Problem, dass das einfache Netzwerk einfach nicht mehr als 100Mbit/s übertragen kann. Wenn man nun von außerhalb versucht das Ganze zu steuern, dann hat man einen noch größeren Geschwindigkeitsverlust außer man ist mit einer T3-Netzwerk-Anbindung und internen 1Gbit/s-Netzwerkschnittstellen oder ähnlichem gesegnet. Die KVMIP-Switches bieten jedoch einen größeren Schutz als die VNC-Lösungen, und das von Hause aus. Denn die KVMIP-Switches unterstützen die neusten SSL-Verschlüsselungen und verschlüsseln ihre Verbindungen auch mit Hilfe dieser das Ganze 1024Bit stark. Teurere Lösungen bieten sogar noch mehr Schutz. Bei den KVMIP-Switches gilt tatsächlich, dass je teurer desto mehr Komfort bzw. intelligentere Lösungen werden geboten.

Die Modelle unterscheiden sich alle sehr stark und man sollte auf die gelieferten Schnittstellen etc. achten. Die Preise gehen von billigen 350 Euro bis in die tausenden. Dabei ist es natürlich

auch abhängig davon, ob es nur die Netzwerk-Erweiterung zum KVM-Switch oder eine eigenständige KVMIP-Lösung ist. Weiterhin ist entscheidend wieviele PCs angeschlossen werden müssen. Wenn nur 4 PCs angeschlossen gehören, dann kann man schon relativ günstig einen KVM-Switch erwerben, wenn es jedoch auf die 16 PCs zugeht, dann wird es auch schnell teurer. Außerdem sind die Lösungen auch sehr unterschiedlich um einen Überblick bei der Schaltung der PCs zu bieten. Die meisten Switches, die ich kenne, sind doch ein bisschen unübersichtlich und Kabelgewirr wäre untertrieben. KVM-Switches gehen in die richtige Richtung. Hier steckt noch eine Menge Potential, welches ausgeschöpft werden kann.

Dazu sicher auch lesenwert [ein Artikel](#) von tomshardware.de.

8. Abschließende Worte

Da ich so eben die Grenze von 69.000 Zeichen überschritten habe, und damit bereits 4 Posts brauche um den Ganzen Text in einen Thread zu fassen, kann ich mir nun auch mehr Zeit lassen ein paar abschließende Worte zu sprechen.

SSH ist für mich die performance-stärkste und sicherste Lösung um Computer fernzuwarten. Doch mit SSH ist man schnell verloren, sobald der fernzuwartene Rechner beim Boot-Vorgang hängen bleibt. SSH wird erst am Ende des Boot-Vorganges initialisiert, und erst etwas später kann auf die Schlüsseldateien in den Benutzer-Verzeichnissen zugegriffen werden. Genau das gleiche Schicksal trifft auch alle VNC-Server.

VNC ist eine bequemere Art und Weise entfernte Computer zu administrieren, jedoch kommt hierbei keine wahre Freude auf aufgrund des starken Geschwindigkeitsverlustes über Netzwerk und Internet. VNC bietet auch keine sichere Verschlüsselung, weshalb es nur im Zusammenhang mit SSH genutzt werden sollte.

KVMIP-Switches bieten dagegen bereits diesen Sicherheitstandard intigriert um sich zumindestens auf die Sicherheit der erzeugten Verbindung verlassen zu können. Doch die Performance ist nicht wirklich prima. Innerhalb des Netzwerkes mit den richtigen Netzwerk-Karten lässt es sich performant steuern. Doch sobald es ins Internet heraus geht, ist es nur noch eine Lösung um die allernötigsten Sachen anzustoßen. Die KVMIP-Switches sind im Gegensatz zu SSH oder VNC nicht darauf angewiesen, dass ein Dienst während des Boot-Vorganges gestartet werden muss. Sie sind hardware-technisch angeschlossen, was deren großer Vorteil ist.

Alles in allem existieren unzählige Remote-Lösungen für Computer dort draußen im Internet und man muss nur die Augen offen halten. Der Grundsatz sollte immer lauten zu erst den Server vollständig zu sichern und ihn dann erst der Gefahr des Internets preis zu geben. Bei ferngewarteten Servern sollte grundsätzlich gelten so wenig Dienste wie möglich aktiv zu halten und das System auf einer Minimal-Konfiguration zu betreiben. Sicherheit für die Verbindung zwischen den Computern ist der zweite Schritt, der befolgt werden sollte, wobei ein bisschen Einfallsreichtum bei Passwörtern, Nutzernamen, Einstellungen und Port-Umleitungen gefragt ist.

Sobald man ungewöhnliche Aktivitäten auf seinen Server entdeckt, sollte man über eine komplette Neuinstallation des Rechners nachdenken. Bei OSX ist es nun noch nicht so verbreitet, aber bei Linux- oder Windows-Systemen ist man schneller eine SPAM- und Hacker-Hilfskraft als man denkt. Dies sollte man seinen Mit-Internet-Nutzern einfach nicht antun. Denn diese quälen sich dann mit infizierten Maschinen im Netz herum. Außerdem kann man durch solch gefährlich installierte Software selbst ins Fadenkreuz der Polizei geraten.

Ich freue mich über Verbesserungen und auch Kritik. Ich weiss, dass dies alles etwas länger wurde, als ich es selber gewünscht habe. Jedoch wollte ich lieber ausführlicher als es zu knapp formulieren. Es ging mir hierbei größtenteils darum das Verständnis derjenigen zu erweitern, die sonst wenig bis gar keine Ahnung von der Materie haben/hatten. Ich hoffe das mir das gelungen ist.

Links und Quellen zum Thema:

SSH:

[UsePAMAuthentication](#)

[MacSSH How-To](#)

[PublicKeys unter Linux/UNIX](#)

[How to change SSH-Port](#)

[O'Reilly SSH How-to](#)

[SSHD-Konfigurationsdatei](#)

[SSH-Helper How-to](#)

[RealVNC KVM over-IP](#)

[KVM over-IP Switches im Vergleich](#)

[Sicherheit schaffen auf Unix-System \(IP-Forwarding\)](#)

PS: ~~ich muss erstmal los... ich werde noch ein paar Bilder nachreichen bei Gelegenheit, doch mir fehlt jetzt gerade die Zeit. Danke fürs Verständnis!~~

PPS: so, wurde alles nochmal komplett überarbeitet. Eine Menge Fehler behoben, die aufgrund

der kurzen Nacht etc. entstanden. Falls jetzt noch inhaltliche Fehler zu finden sind, dann sagt doch bitte bescheid. Danke!

iBook G4, 1.25Ghz, OSX 10.4, 80GB, 768MB

[Debian PC-Server](#)

[iPod mini 2.G](#)

Dell Widescreen 2005FPW

Geändert von Cyrics (Heute um 02:39 Uhr).

