Netzwerk

- STUN-Server einrichten
- Dienst läßt sich nicht starten wegen bereits geöffnetem Port - could not bind
- Nicht PXE-fähige Server trotzdem via PXE-Installserver installieren
- TCP-Verbindungen über HTTP tunneln
- Kleine WLAN-Accesspoint Infosammlung
- telnet-Verbindungsversuch beenden
- Spotify an entfernten MPD streamen
- eine Datei per nc (netcat) kopieren
- Traffic-Shaping für ausgehende Mails
- TCP-Stack Tuning für Linux-Server
- FTP
    - vsftpd - FTP-Server anonymous only
    - Automatische Dateiübertragung via FTP
    - .netrc FTP

- tcpdump cheat sheet

STUN-Server einrichten

Das STUN-Protokoll definiert folgendes (http://www.ietf.org/rfc/rfc3489.txt):

- STUN ermöglicht einem Gerät dessen öffentliche IP und den Typ von NAT-Gerät hinter dem es hängt herauszufinden
- STUN arbeitet auf TCP und UDP Port 3478
- STUN ist inzwischen recht verbreitet und wird von vielen VOIP-Geräten unterstützt
- STUN kann DNS SRV-Einträge verwenden um STUN server einer bestimmten Domain herauszufinden. Der Servicename ist „_stun._udp or _stun._tcp“

1. Download des Sourcecodes nach /usr/src von:

```
http://sourceforge.net/projects/stun/
```

2. Entpacken und Kompilieren der Sourcen:

```
cd /usr/src
tar xvfz stund_0.96_Aug13.tgz
cd stund
make
mkdir /usr/local/stund
cp server /usr/local/stund
cp client /usr/local/stund
```

3. STUN-Server testweise starten:

```
cd /usr/local/stund
./server -v -h <main IP> -a <secondary IP>
```

4. STUN-Server testen (neues Konsolenfenster aufmachen):

```
cd /usr/local/stund
./client -v <main IP>
```

5. Wenn der STUN-Server antwortet ist alles OK

6. Hier noch ein Redhat kompatibles init-Script für den Server

```
#!/bin/sh
#
#   Startup/shutdown script for the STUN-Server
#
```

```
#   Linux chkconfig stuff:
#
#   chkconfig: 35 90 10
#   description: Startup/shutdown script for the STUN-Server
#

# Source function library.
if [ -f /etc/init.d/functions ] ; then
. /etc/init.d/functions
elif [ -f /etc/rc.d/init.d/functions ] ; then
. /etc/rc.d/init.d/functions
else
exit 0
fi

DAEMON=/usr/local/stund/server
IP1="86.109.254.36"
IP2="86.109.254.37"

prog=stund

start () {
echo -n $"Starting $prog: "

# start daemon
daemon $DAEMON -b -h $IP1 -a $IP2
        RETVAL=$?
echo
[ $RETVAL = 0 ] && touch /var/lock/subsys/stund
return $RETVAL
}

stop () {
# stop daemon
echo -n $"Stopping $prog: "
killproc $DAEMON
RETVAL=$?
echo
[ $RETVAL = 0 ] && rm -f /var/lock/subsys/stund
}
```

```
restart() {
	stop
	start
}

case $1 in
	start)
		start
	;;
	stop)
		stop
	;;
	restart)
		restart
	;;
	condrestart)
		[ -f /var/lock/subsys/stund ] && restart || :
	;;
	status)
		status $DAEMON
	;;
	*)

	echo $"Usage: $prog {start|stop|restart|condrestart|status}"
	exit 1
esac

exit $RETVAL
```

Dienst läßt sich nicht starten wegen bereits geöffnetem Port - could not bind

Beispiel: Folgender Fehler beim Starten des Apache:

```
# service httpd restart
Starting httpd: (98)Address already in use: make_sock: could not bind to address 0.0.0.0:443
no listening sockets available, shutting down
Unable to open logs!
```

Nun muss geprüft werden, welche Prozesse die Port bereits geöffnet haben, in diesem Fall Port 443 und wahrscheinlich auch Port 80. Dazu gibts verschiedenste Möglichkeiten:

```
1. netstat -lnp | grep '0.0.0.0:443'
2. lsof | grep *:https
3. lsof -i:443
4. fuser -v 443/tcp
5. netstat -tulpen | grep :443
```

Jetzt kann man diese Prozesse entsprechend killen und dann nochmal versuchen seinen Dienst zu starten.

Nicht PXE-fähige Server trotzdem via PXE-Installserver installieren

Server von Boot-CD des gewünschten OS starten (im Beispiel Centos4.4) und am ersten Bootbildschirm folgendes eingeben:

```
linux ksdevice=eth0 ip=dhcp network ks=nfs:pxeserver.meinserver.de:/var/tftpboot/linux-install/c
```

TCP-Verbindungen über HTTP tunneln

Mit dem kleinen Tool httptunnel lassen sich TCP-Verbindungen über das HTTP-Protokoll tunneln. Dies ist dann z.B. interessant, wenn man hinter einer Firewall sitzt, die nur Verbindungen über Port 80 nach draussen läßt.

Benötigt wird ein Server außerhalb, z.B. der Server zuhause oder im Internet.

In diesem Beispiel will ich eine Verbindung zu einem SSH-Server im Internet aufbauen. Der kann aber muss nicht auf dem Zielserver laufen. In diesem Fall will ich die Verbindung auf einen anderen Server weiterleiten.

Zuerst wird auf dem Ziel der Server für httptunnel gestartet:

```
Syntax:
hts -F <weiterleitung an ip>:<zielport> <lokaler port>
Beispiel:
hts -F meinssh.server.de:22 1234
oder
hts -F localhost:22 1234
```

danach auf dem Quellserver der Client:

```
Syntax:
htc -F <lokaler port> <httptunnel server>:<httptunnel port>
Beispiel:
htc -F 5222 mein-httptunnel-server.de:1234
```

Sobald beide Programme gestartet sind, sollte der Tunnel stehen. Jetzt kann ich mich mit „ssh localhost -p5522“ über den Tunnel auf meinen Zielserver verbinden.

Kleine WLAN-Accesspoint Infosammlung

Verfügbare WLAN-Netzwerke scannen:

```
iwlist ath0 scan
wlanconfig ath0 list ap
```

telnet-Verbindungsversuch beenden

jeder Admin kennt es: Um schnell mal eine TCP-Verbindung zu testen ist telnet gut geeignet. Was tun wenn man telnet (wie etwa bei SMTP) nicht über quit oder exit beenden kann?

Hier hilft der sogenannte Escape-Character, der bei jedem Verbindungsaufbau angezeigt wird:

```
[root@bla ~]# telnet testserver.de 636
Trying 192.168.70.245...
Connected to testserver.de (192.168.70.245).
Escape character is '^]'.
```

Einfach die angezeigte Taste drücken und man landet an der Telnet-Konsole, die man über „quit“ verlassen kann:

```
^]
telnet> quit
Connection closed.
```

Bei deutschem Tastaturlayout sind das die Tasten: <Strg> + <AltGr> + 9 Bei englischem Layout: <Strg> + <+>

Spotify an entfernten MPD streamen

Die Audio-Ausgabe von Spotify läßt sich mit einfachen Mitteln an einen MPD-Server (oder sonstigen Stream-Dienst) senden. Für Spotify unter Linux verwende ich die Wine-Version.

Vorraussetzung ist das Paket libavcodec-extra-53 zur MP3-Dekodierung und vlc (VideoLAN Client). Am einfachsten läßt es sich über dieses Kommando installieren (Xubuntu/Ubuntu):

```
sudo apt-get install ubuntu-restricted-extras
```

VLC sagen wir, dass er als Quelle den Pulse-Audio Server nehmen soll und zur Ausgabe einen HTTP-Stream im OGG-Format auf Port 8000 bereitstellen soll.

```
cvlc pulse:// --sout '#transcode{acodec=mp3,ab=256}:std{access=http,mux=ogg,dst=<deine-LAN-IP>:8
```

Im Pulse-Audio Control Tool stellen wir folgendes ein: pavucontrol ? Recording ? VLC auf „Monitor of <deine Soundkarte>“

Beim entfernten MPD-Server kann jetzt der Stream abgegriffen werden. Entweder über eine Oberfläche wie GMPC oder per Konsole diesen Stream hinzufügen:

```
http://<deine-LAN-IP>:8000
```

Jetzt Spotify starten und einen Song abspielen. MPD sollte nun das Audio ausgeben. Durch die Pufferung kommt es hier zu einer kleinen Verzögerung.

eine Datei per nc (netcat) kopieren

Dateien per SCP zu kopieren kann manchmal etwas anstregend sein (z.B. weil der Benutzer kein Passwort konfiguriert hat, SSH-Agent Forwarding deaktiviert ist (weil böse) oder man keinen Key einrichten will).

Ein einfacher Weg um eine Datei auf einen anderen Host zu kopieren ist das Tool nc (netcat).

Dazu wird das Programm auf dem Zielhost im Listener Modus gestartet und die Standardausgabe an tar gepiped:

```
nc -l -p 2000 | tar x
```

auf dem Quell-Host wird die Datei durch tar geschickt und an netcat gepiped, das diese wiederrum an den Zielserver schickt (klappt auch für mehrere Dateien gleichzeitig):

```
tar -cf - <FILE> | nc <TARGET> 2000
```

Leider gibts dabei weder eine Fortschrittsanzeige, noch sieht man wirklich wenn die Übertragung beendet wurde. Das läßt sich jedoch einfach per md5sum prüfen. Wenn die Summen gleich sind, kann man das Programm abbrechen.

Alternativ (tar mit z Parameter komprimiert die Daten):

```
# Zielhost:
$ nc -l -p 2000 | tar xz

# Quellhost:
$ tar -czf - <ordner> | nc <zielserver> 2000
```

Traffic-Shaping für ausgehende Mails

Das folgende Snippet ist nützlich wenn man beispielsweise nur einen begrenzte Upload-Bandbreite verfügt (z.B. 1mbit). Damit läßt sich ausgehender Traffic mit Zielport 25 limitieren, sodass anderer Traffic uneingeschränkt funktioniert (etwa SSH-Sessions).

Es muss beim Systemstart ausgeführt werden, daher z.B. in /etc/rc.local eintragen:

```
tc qdisc add dev eth0 root handle 1: htb default 30
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 500kbit ceil 500kbit burst 15k
tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip dport 25 0xffff flowid 1:20
```

TCP-Stack Tuning für Linux-Server

Dies ist meine aktuelle sysctl Konfiguration für Linux-Server. Es wird nicht nur der TCP-Stack behandelt, sondern auch Parameter die Kernel und Dateisysteme betreffen.

Nach Änderungen `sysctl –system` ausführen:

```
fs.file-max = 100000
fs.inotify.max_user_watches = 524288
fs.protected_hardlinks = 1
fs.protected_symlinks = 1

kernel.kptr_restrict = 1
kernel.printk = 4 4 1 7
kernel.sysrq = 176
kernel.yama.ptrace_scope = 1

net.core.default_qdisc=fq
net.core.netdev_max_backlog = 30000
net.core.rmem_default = 26214400
net.core.rmem_max = 26214400
net.core.wmem_max = 26214400

net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.ip_local_port_range = 1025 65535
net.ipv4.tcp_congestion_control=bbr
net.ipv4.tcp_fin_timeout = 3
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_mtu_probing = 0
net.ipv4.tcp_no_metrics_save = 1
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_wmem = 4096 65536 16777216

net.ipv6.conf.all.use_tempaddr = 0
net.ipv6.conf.default.use_tempaddr = 0
net.ipv6.conf.eth0.use_tempaddr = 0
net.ipv6.conf.eth1.use_tempaddr = 0
net.ipv6.conf.lo.use_tempaddr = 0

vm.mmap_min_addr = 65536
vm.swappiness = 1
```

FTP

alt, aber manchmal nicht vermeidbar

FTP

vsftpd - FTP-Server anonymous only

```
#
# very simple anonymous FTP server configuration
#
# Mandatory directives
#
listen=YES
local_enable=NO
anonymous_enable=YES
write_enable=NO
anon_root=/srv/ftp
#
# Optional directives
#
anon_max_rate=2048000
xferlog_enable=YES
listen_address=123.123.123.123
listen_port=21
```

FTP

Automatische Dateiübertragung via FTP

um automatische via Script FTP-Übertragungen zu steuern gibts u.a. folgende Möglichkeiten:

- FTP kann via automatischer_ftp-login_.netrc Datei (im Home des entsprechenden Users abgelegt) gesteuert werden. Weitere Infos liefer die manpage zu netrc
- die andere und, meiner Meinung nach, elegantere Lösung via sogenanntem Here-Dokument. Hier wird Text an die Standardeingabe eines Kommandos geschickt, wobei auch Variablen verwendet werden können:

```
#!/bin/bash
# upload.sh
DATE=`date "+%Y-%m-%d"`
local=/tmp/bild1.jpg
remote=bild1-$DATE.jpg

ftp -n <<EOFTP
  open ftp.ftpserver.de
  user backup passwort
  bin
  put $local $remote
  quit
EOFTP
```

.netrc FTP

FTP login automatisieren

Wenn man regelmäßig FTP verwendet - beispielsweise um seine Homepage bei einem Provider zu aktualisieren – ist es praktisch den Login Prozess zu automatisieren, sodaß es nicht länger notwendig ist, jedesmal den Benutzernamen und das Passwort einzugeben.

Das FTP Programm sucht im Homeverzeichniss des Anwenders nach der Datei „.netrc“. In dieser Datei kann man den Namen von Rechnen zusammen mit einem Username und dem zugehörigen Passwort eintragen. Wählt man später einen der eingetragenen Rechner an, loggt sich das FTP Programm automatisch ein. Will man sich unter einen anderen Account einloggen, kann man über den Parameter „-n“ die Automatik abschalten.

Wegen der offensichtlichen Sicherheitsproblematik, die durch das hinterlegen von Passwörtern in einer Datei entsteht, weigert sich das FTP Programm die Datei auszuwerten, wenn sie nicht die richtigen Zugriffsrechte besitzt. Die Datei darf nur vom Anwender gelesen werden, also den Modus 600 haben. Das Kommando

```
chmod 600 .netrc
```

setzt diesen richtigen Modus. Für eine FTP-Session mit einem automatischen anmelden beim Rechner ‘ftp.upload.com’ müßte die Datei .netrc folgende Zeile beinhalten.

```
machine ftp.upload.com name UserName password PasswortFuerUserName
```

tcpdump cheat sheet

Tcpdump is a commandline tool that is used to dump traffic on a network. This tool comes in hand when you want to analyse network captures within the command line. Basically it can do most of the wireshark job.

Additional Note & Reference

To be fair: This gist is itself a fork I created some time ago, but the original gist or author seems to not exist anymore, and it looks like that I'm now in the lead ;-) Please see the revision history for details.

Furthermore some more and basic advanced examples may be of interest (thanks to twitter://@howtouselinux1):

- Tcpdump Cheat Sheet (Basic Advanced Examples)

Options

The following are some of options that I prefer when using `tcpdump` for my daily use.

tcpdump [OPTIONS]

```
-i any : Listen to all the interfaces
-i virbr0: Listen to a specific interface virbr0
-D: Show the list of available interface
-n: Don't resolve the hostnames
-nn: Don't resolve hostnames or port names.
-q: quite output
-t: Don't print a timestamp on each dump line.
-tttt: Give maximally human-readbale timestamp output
-X: Show the packet's contents in both HEX ad ASCII
-XX: Same as -X but shows the ethernet header.
-v, -vv, -vvv: Being more verbose(increase number of packet information)
-c: Only capture number of packets and stop
-s: Define the snaplength(size) of the capture in bytes. Use -s0 to get
everything.
-S: Print absolute sequence numbers.
-e: Get the ethernet header as well
-E: Decrypt IPSEC traffic by providing an encryption key.
```

Expressions

`tcpdump` allow us to use expression so we can narrow down our solution to get exactly what we're looking for.

There are 3 types of expression: `type`, `dir` and `proto`

- Type options are: `host`, `net`, and `port`
- Direction are: `src` and `dst`
- Protocol : `tcp`, `udp`, `icmp`, `ah` etc

Examples

1. Basic communication to see what happens on the network

```
$ tcpdump -i any
```

2. Monitor specific interface

```
$ tcpdump -i virbr0
```

3. Raw output view with verbose output,no host/port resolution,absolute sequence number and human-readable timestamps.

```
$ tcpdump -ttttnnvvS
```

4. Find traffic by IP

```
$ tcpdump host 192.168.122.131
```

5. Seeing packets with HEX output

```
$ tcpdump -nnvXSs 0 -c1 icmp
```

6. Filtering by Source and Destination

```
$ tcpdump src 192.168.122.131
$ tcpdump dst 192.168.122.14
```

7. Finding packets by network

```
$ tcpdump net 192.168.122.0/24
```

8. Show traffic related to a specific port

```
$ tcpdump port 3389
```

9. Show traffic of one protocol

```
$ tcpdump icmp
```

10. Show only IPv6 Traffic

```
$ tcpdump ip6
```

11. Find traffic using Port ranges

```
$ tcpdump portrange 21-25
```

12. Find traffic base on packet size

```
$ tcpdump less 32


$ tcpdump greater 32


$ tcpdump <= 102
```

13. Writing captures to a file

```
$ tcpdump port 80 -w output
```

14. Reading from pcap files

```
$ tcpdump -r output.pcap
```

More Examples

1. Options Combination
   - AND : `and` or `&&`
   - OR : `or` or `||`
   - EXCEPT : `not` or `!`

```
$ tcpdump -nnvvS src 192.168.122.1 and dst port 4444
```

2. Complex grouping and special characters For complex grouping we use `()` to specify our options

```
$ tcpdump 'src 192.168.122.84 and (dst port 4444 or 22)'
```

3. Isolating Specific TCP Flags. The filter `tcp[13]` look at offset 13 in `TCP HEADER`,hence the number represent the location within the byte, while the `!=0` means that the flag is set to 1.
   Show all URGENT (URG) packets\

```
$ tcpdump 'tcp[13] & 32!=0'
```

   Show all ACKNOWLEDGE(ACK) packets\

```
$ tcpdump 'tcp[13] & 16!=0'
```

Show all PUSH (PSH) packets\

```
$ tcpdump 'tcp[13] & 8!=0'
```

Show all RESET (RST packets\

```
$ tcpdump 'tcp[13] & 4!=0'
```

Show all SYNCHRONIZE (SYN) packets\

```
$ tcpdump 'tcp[13] & 2!=0'
```

Show all FINISH (FIN) packets\

```
$ tcpdump 'tcp[13] & 1!=0'
```

Show all SYNCRONIZE/ACKNOWLEDGE (SYNACK) packets\

```
$ tcpdump 'tcp[13]=18'
```

Alternative we could also use `tcpflags` syntax

```
$ tcpdump 'tcp[tcpflags] == tcp-syn'
$ tcpdump 'tcp[tcpflags] == tcp-rst'
$ tcpdump 'tcp[tcpflags] == tcp-fin'
```

4. Identifying malformed/malicious packets.
   - Packets with both rst and syn flags shouldn't be the case.

     ```
     $ tcpdump 'tcp=[13] = 6'
     ```

   - Find cleartext http get requests

     ```
     $ tcpdump 'tcp[32:4] = 0x47455420'
     ```

   - Find ssh connection on any port via (banner text)

     ```
     $ # tcpdump 'tcp[(tcp[12]>>2):4] = 0x5353482D'
     ```