netcat

Johannes Franken
<jfranken@jfranken.de>

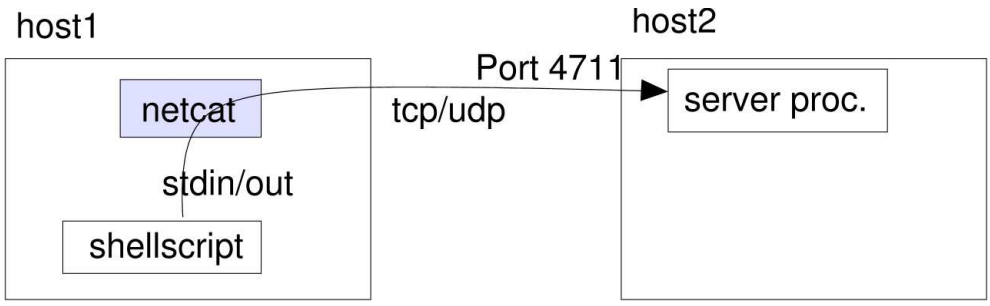
Auf dieser Seite zeige ich Anwendungsbeispiele für **`netcat`**, ein Kommandozeilentool zum Erstellen von Netzverbindungen über tcp oder udp.

Inhalt

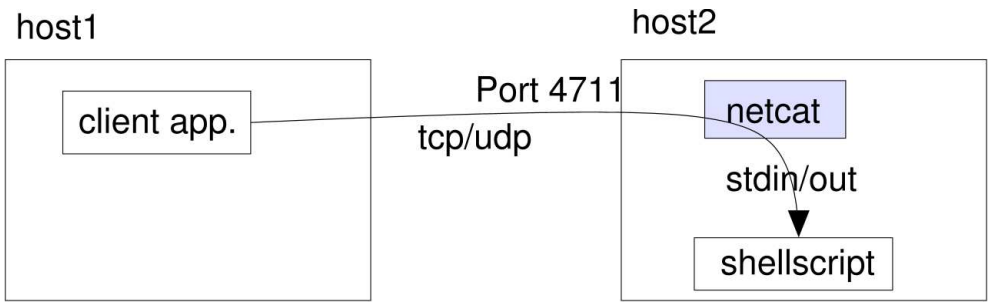
Verwendungszweck

`netcat` ermöglicht die Kommunikation mit einer tcp- oder udp-Verbindung über stdin/out, beispielsweise aus einem Shellscript heraus.

`netcat` kann die Verbindung wahlweise selbst herstellen (Client-Modus):



oder entgegennehmen (Server-Modus):



Zusätzlich kann **`netcat`** im Clientmodus als Portscanner arbeiten.

Die folgende Dokumentation bezieht sich auf die bis heute (2003) aktuelle Version 1.10 vom 20.03.1996.

Aufruf als Client

Webserver abfragen

Zum Vergleich: Bei dem Versuch, einen Webserver mittels Shellumleitung abzufragen, wartet telnet nicht auf die Antwort des Servers, sondern bricht die Verbindung schon am Ende der Eingabe ab:

```
$ printf 'GET / HTTP/1.0\n\n'  | telnet www.jfranken.de 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.
$
```

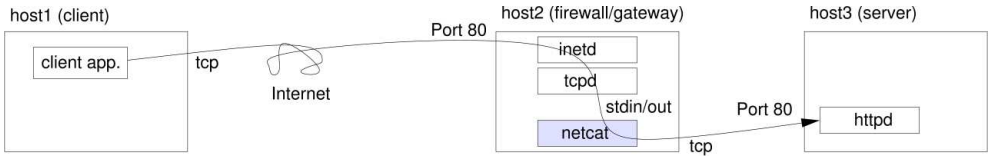
Mit **`netcat`** klappt das besser:

```
$ printf 'GET / HTTP/1.0\n\n'  | nc -w 10 www.jfranken.de 80
HTTP/1.0 200 OK
Server: thttpd/2.21b 23apr2001
Content-Type: text/html; charset=iso-8859-1
[...]
<HTML>
[...]
</HTML>
```

`netcat` beendet sich erst, wenn die Verbindung abbricht. Der Parameter **`-w 10`** bewirkt übrigens, dass dies spätestens 10 Sekunden nach dem Ende der Eingabe der Fall ist.

Ports umleiten

Mit **`netcat`** und inetd kann ich lokale Ports an andere Hosts umleiten.



Die folgende Zeile in **`/etc/inetd.conf`** auf host2 stellt den Port 80 des host3 lokal zur Verfügung:

```
80 stream tcp nowait nobody /usr/bin/nc /usr/bin/nc -w 3 host3 80
```

Wer Wert auf Protokollierung und Zugriffskontrolle legt, kann hier das erste **`/usr/bin/nc`** durch **`/usr/sbin/tcpd`** ersetzen.

Mailrelays ansprechen

In meinem Vortrag über SSH-Tunnelling zeige ich eine exim-Konfiguration, die Mails an gewisse Domains über ein Mailrelay leitet, das per ssh und **`netcat`** angesprochen wird:

```
# at bottom of TRANSPORT CONFIGURATION : ############################

# Transport, which tunnels mails over ssh to my smarthost
ssh:
  driver = pipe

  # talk regular smtp to the pipe (qmail approved :-) )
  bsmtp = all
  bsmtp_helo = true
  use_crlf = true
  prefix = "
  suffix = "

  # connect to my mail server's smtp port.
  # must authenticate without password
  command = ssh johannes.our-isp.org netcat -w 1 localhost smtp
  user = jfranken

  # ssh failure handling: kill hanging sessions,
  # retry and tell me on failures
  timeout = 300s
  temp_errors = 1
  return_fail_output = true

# at top of ROUTERS CONFIGURATION: ############################
t_online:
  driver = domainlist
  transport = ssh
  route_list = "^t-online\.(de|at|com)|t-dialin\.net$ ;\
                ^mail.com$                           ;\
                ^lists.sourceforge.net$              ;\
                ^bigpond.com$                        ;\
                ^aol.com$                            ;\
                ^mathematik.tu-darmstadt.de$         ;"
```

Das Interessante bei dieser Konfiguration ist die Erkenntnis, dass ich jeden Rechner, auf den ich per ssh zugreifen kann, als Mailrelay verwenden kann.

Die Software eines DNS-Servers feststellen

Man kann "von außen" feststellen, auf welcher Software ein entfernter DNS-Server läuft:

```
$ dig @pns.dtag.de version.bind. txt CHAOS|grep -i "^V"
VERSION.BIND.           0S CHAOS TXT    "BIND 8.3.4"
```

Das geht natürlich auch mit bash und **`netcat`**:

```
$ whatdns() {
   printf 'begin-base64 644 -\np8IBAAABAAAAAAAAB3ZlcnNpb24EYmluZAAAEAADCg==\n====’|
   uudecode| nc -uw 1 $1 domain | strings| tail -1; }
$ whatdns pns.dtag.de
BIND 8.3.4
$ whatdns 141.2.1.1
4.9.7
```

Aufruf als Server

Wenn ich

```
PROGRAM | nc -l -p PORT -w TIMEOUT
```

oder

```
nc -l -p PORT -w TIMEOUT | PROGRAM
```

aufrufe,

1. erwartet **netcat** genau eine Verbindung auf Port **PORT**,
2. verbindet diese mit der Aus- oder Eingabe von **PROGRAM** und
3. wartet maximal **TIMEOUT** Sekunden darauf, dass die Verbindung abgebrochen wird.

Für Ports unter 1024 muss ich root sein.

Wenn der Port bidirektional sein (also sowohl ein- als auch ausgeben) soll, rufe ich

```
nc -l -p PORT -e PROGRAM
```

auf. Leider kann **netcat** keine Parameter an PROGRAM übergeben, daher wird man hier häufig ein 2-Zeilen Shellscript angeben.

Da sich **netcat** beim Ende der Verbindung beendet, kann es nur eine einzige Verbindung bedienen. Wer eine Lösung sucht, bei der **netcat** mehrere Verbindungen (nacheinander oder gleichzeitig) entgegennimmt, kann das folgende, 3-zeilige Shellscript verwenden:

```
$ cat <<'EOF'>mydemon
#!/bin/bash
export port=${port:-$1} # inherit $1 as $port
nc -l -p $port -e $0 &  # await further connections on this port
[ $1 ]|| PROGRAM        # do the work (not for the first invocation)
EOF
$ chmod +x mydemon
$ ./mydemon 3000
$
```

Der Parameter **-e** funktioniert übrigens nur, wenn **netcat** mit der Option **-DGAPING_SECURITY_HOLE** compiliert worden war. Bei Debian GNU/Linux 3.0 ist das der Fall.

Remote shell Dämon

Mit folgendem Shellscript starte ich mir einen persönlichen Remote shell daemon:

```
#!/bin/bash
# pershd - personal shell demon
# 2003-02-17 Johannes Franken  - jfranken@jfranken.de

# on first invocation, export $PW and $port to subshells
export PW port
if [ $1 ] ; then
        port=$1
        echo -n pass:
        read PW
fi

# do we know the port to listen on?
if [ ! $port ]; then
        echo USAGE: $0 port ;
        exit;
fi

# wait for further connections (veiling params)
echo "-l -p $port -e $0" | nc 2>/dev/null &
[ $1 ] && exit; # first invocation exit here

# ask for password
unset p
until [ "$p" = "$PW" ]; do
        echo -n pass:
        read p
done

# received good password. present a shell
bash --noediting -i
```

Wenn ich dieses Script aufrufe und ein Passwort vergebe, legt es sich in den Hintergrund und wartet auf Verbindungen:

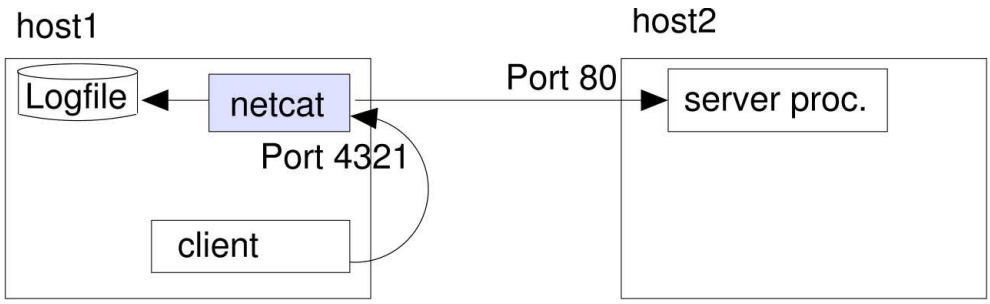
```
jfranken@tp:~$ ./pershd 1234
pass:secret
jfranken@tp:~$
```

Mit diesem Passwort kann ich von anderen Rechnern aus auf meine Shell zugreifen:

```
$ nc tp.jfranken.de 1234
pass:secret
jfranken@tp:~$
jfranken@tp:~$ exit
exit
$
```

Netzverkehr mitschneiden

Sniffer wie **`tcpdump`**, **`snoop`**, **`iptraf`** oder **`ethereal`** sind beliebte Werkzeuge zum Debuggen von Netzanwendungen, erfordern jedoch Rootrechte. Auch als User kann ich die Kommunikation mitschneiden, indem ich den Client auf einen lokalen **`netcat`** richte, der alles an den Server weiterleitet und dabei loggt.



Hierzu ersetze ich in o.g. mydemon-Script **PROGRAM** durch

```
netcat -o /tmp/sniffer.‘date +%s.$$‘ tp.jfranken.de 80
```

und rufe es mit Parameter 4321 auf. Für jeden Zugriff auf diesen Port erstellt **netcat** jetzt eine Logdatei unter **/tmp**.

Ein Beispiel: Nachdem ich in meinem Browser **http://localhost:4321** aufgerufen habe, entsteht folgende Logdatei:

```
$ cat /tmp/sniffer.1045474262.5864
> 00000000 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a # GET / HTTP/1.1..
> 00000010 55 73 65 72 2d 41 67 65 6e 74 3a 20 4f 70 65 72 # User-Agent: Oper
> 00000020 61 2f 36 2e 31 31 20 28 4c 69 6e 75 78 20 32 2e # a/6.11 (Linux 2.
[...]
< 00000000 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d # HTTP/1.1 200 OK.
< 00000010 0a 53 65 72 76 65 72 3a 20 74 68 74 74 70 64 2f # .Server: thttpd/
< 00000020 32 2e 32 31 62 20 32 33 61 70 72 32 30 30 31 0d # 2.21b 23apr2001.
< 00000030 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 74 # .Content-Type: t
[...]
< 00002180 0a 3c 2f 46 52 41 4d 45 53 45 54 3e 0a 3c 2f 48 # .</FRAMESET>.</H
< 00002190 54 4d 4c 3e 0a                                  # TML>.
```

Verzeichnisbäume über ein Internet kopieren

Mit Hilfe von von **netcat**, (gnu)tar und bash kann ich Verzeichnisbäume von einem Host auf einen anderen kopieren. Die Dateien werden komprimiert und inkl. ihrer Attribute (Zugriffsrechte/Änderungsdatum) übertragen.

Hierzu lasse ich auf dem empfangenden Host einen **netcat** auf Port 51330 lauschen und leite seine Ausgabe in ein **tar** um:

```
$ alias receive=’nc -vlp 51330 | tar xzvp’
$ receive
listening on [any] 51330 ...
```

Auf der sendenden Seite rufe ich eine Shellfunktion auf, welche die übergebenen Dateien/Verzeichnisse zusammentart und an den Port 51330 des empfangenden Rechners sendet:

```
$ send() { j=$*; tar cpz ${j/%${!#}/}|nc -w 1 ${!#} 51330;}
$ send dir* tp.jfranken.de
```

Zur Bash-o-magic:

- **${!#}** gibt den letzten Parameter (den Hostnamen) zurück,
- **${j/%${!#}/}** die übrigen (Liste der Dateien oder Verzeichnisse).

Aufruf als Portscanner

Mit dem folgenden Aufruf ertestet **`netcat`** die offenen Ports eines Hosts:

```
nc -vz PARAMETER hostname PORTRANGES
```

Mögliche **PARAMETER** sind:

Parameter	Bedeutung
`-v`	auch die geschlossenen Ports anzeigen
`-w` <Sekunden>	Timeout für Verbindungsaufbau
`-u`	udp statt tcp. Am besten zusammen verwenden mit **`-w`**
`-r`	Die Ports in zufälliger Reihenfolge ansprechen
`-i` <Sekunden>	Pause zwischen den Verbindungsversuchen

Beispiele für einen TCP- und einen udp-Scan:

```
$ nc -vz www.jfranken.de 1-1024
www.jfranken.de [195.88.176.20] 443 (https) open
www.jfranken.de [195.88.176.20] 110 (pop3) open
www.jfranken.de [195.88.176.20] 80 (www) open
www.jfranken.de [195.88.176.20] 53 (domain) open
www.jfranken.de [195.88.176.20] 25 (smtp) open
www.jfranken.de [195.88.176.20] 22 (ssh) open
$
$ nc -vz -vur -i 1 -w 2 localhost 1024 108-112
localhost [127.0.0.1] 1024 (?) open
localhost [127.0.0.1] 109 (pop2) : Connection refused
localhost [127.0.0.1] 112 (?) : Connection refused
localhost [127.0.0.1] 111 (sunrpc) open
localhost [127.0.0.1] 110 (pop3) : Connection refused
localhost [127.0.0.1] 108 (?) : Connection refused
 sent 0, rcvd 0
```

Weiterführende Links

- Die netcat(1) manpage
- Die netcat README Datei