

# netcat

Johannes Franken  
<jfranken@jfranken.de>

On this page I show example uses of `netcat` - a command line tool to create network connections over tcp or udp.

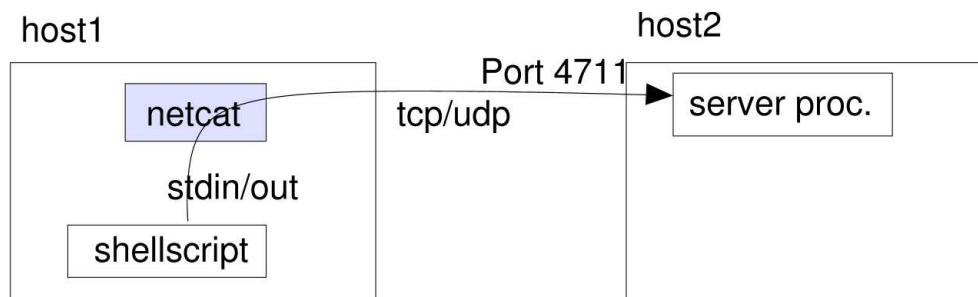
## Contents

1. [Intended purpose](#)
2. [Usage as client](#)
  - a) [Querying webservers](#)
  - b) [Forwarding ports](#)
  - c) [Talking to mailrelays](#)
  - d) [Determining a remote DNS server's software](#)
3. [Usage as server](#)
  - a) [Remote shell demon](#)
  - b) [Capturing network traffic](#)
  - c) [Copying directory trees over some internet](#)
4. [Usage as portscanner](#)
5. [Links to advanced topics](#)

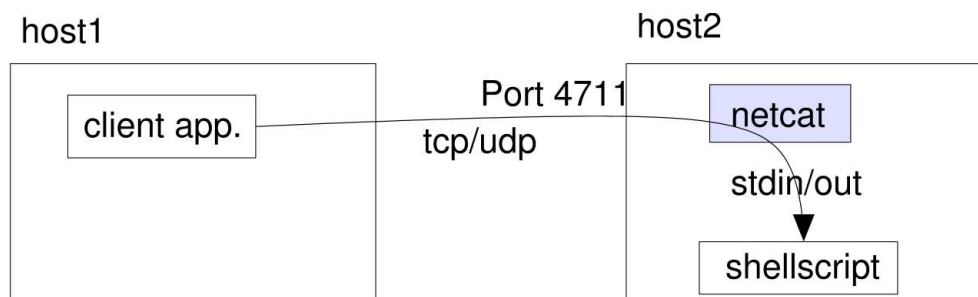
## Intended purpose

With `netcat` you can talk to arbitrary tcp- or udp-connections via stdin/out, e.g. from a shellsript.

`netcat` can either setup the connections itself (client-mode):



or answer them (server-mode):



Additionally `netcat`'s client-mode can be used as a portscanner.

The following documentation correlates to `netcat` 1.10 from 1996-03-20, which still is the most up to date version today (2003).



# Usage as client

## Querying webservers

Try it yourself: When you query a webserver using shell redirection, telnet won't wait for the server's response, but abort the connection at the end of it's input:

```
$ 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.
$
```

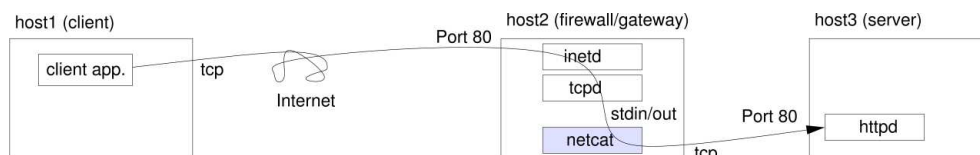
**netcat** performs better:

```
$ 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** will quit on the time when the connection closes down. the BTW: The parameter `-w 10` makes sure, that this happens maximum 10 seconds the after the the input has ended.

## Forwarding ports

With **netcat** and **inetd** you can forward local ports to other hosts.



The following line in `/etc/inetd.conf` on host2 will locally provide host 3's port 80.

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

If you like logging and access control, you can change the first `/usr/bin/nc` to `/usr/sbin/tcpd`.

## Talking to mailrelays

In my [talk about ssh-tunnelling](#) you can see a config file for **exim**, which forwards mails to certain domains over a mail relay using **ssh** and **netcat**:

```

# 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 :- )
  bsmtplib = all
  bsmtplib_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$ ;"

```

The interesting point about this configuration is the cognition, that I can use any host, for which I have ssh access, as a mail relay.

## Determining a remote DNS server's software

You can remotely determine the software a remote DNS server is running on:

```

$ dig @pns.dtag.de version.bind.txt CHAOS|grep -i "^v"
VERSION.BIND.          OS CHAOS TXT          "BIND 8.3.4"

```

Of course, you can also do that with bash and netcat:

```

$ whatdns() {
  printf 'begin-base64 644 -\np8IBAAABAAAAAAB3ZlcnNpb24EYmluZAAAEEAADc==\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

```

# Usage as server

When you type

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

or

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

1. **netcat** will wait for exactly one connection to port **PORT**, then
2. attach to **PROGRAM**'s out- or input and
3. wait maximum **TIMEOUT** seconds for the connection to abort.

For ports below 1024, you need to be root.

If the port should be bidirectional (do input as well as output), you can type

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

Unfortunately **netcat** cannot pass parameters to **PROGRAM**, so you might need to supply a 2-lines shellscrip in most cases.

Because **netcat** quits at the end of the connection, it can only service one connection. Use the following 3-lines shellscrip, if you want **netcat** to accept multiple (successive or concurrent) sessions:

```
$ 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
$
```

The **-e** parameter will only work, if **netcat** has been compiled with the option **-DGAPING\_SECURITY\_HOLE** set. For Debian GNU/Linux 3.0, this is true.

## Remote shell demon

With the following shellscrip, you can start your personal remote shell demon:

```
#!/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
```

When I start this script, it asks me to tell it a password and then waits for connections in the background:

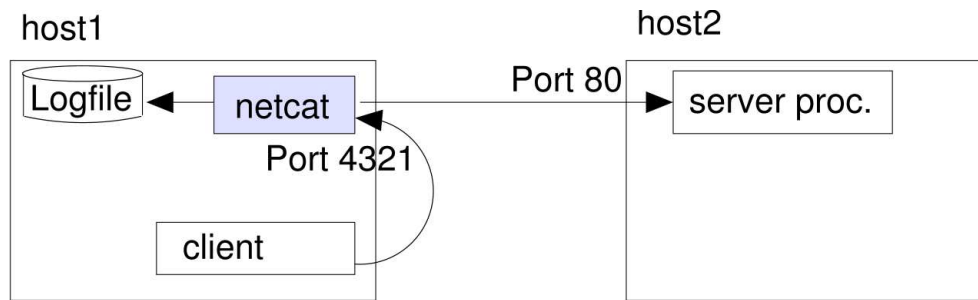
```
jfranken@tp:~$ ./pershd 1234
pass:secret
jfranken@tp:~$
```

With this password I can now access my shell from remote hosts:

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

## Capturing network traffic

Sniffing tools like `tcpdump`, `snoop`, `iptraf` or `ethereal` are quite popular for debugging network applications. But in order to use them, you need to be root. With `netcat` you can also capture network connections as a normal user: just point your application to a local netcat, which logs everything and then forwards it to the server.



To do so, just replace the term **PROGRAM** in the above [mydemon-script](#) with

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

and call it with parameter 4321. netcat will now create a logfile for each access to that port in your /tmp-directory.

For example: After I had opened <http://localhost:4321> with my browser, there was the following logfile:

```
$ 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>.
```

## Copying directory trees over some internet

With netcat, (gnu)tar and bash, you can copy directory trees from one host to another. The files are transferred compressed and with all their attributes (permissions/modification times).

To receive a file, I let netcat listen on port 51330, and redirect it's output to a tar:

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

On the sending host you call a shell function, which tars the given files/directories and sends them to port 51330 at the receiving host.

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

For those not familiar with bash-o-magic:

- `${!#}` returns the last parameter (the hostname),
- `${j/%${!#}/}` any others (list of files or directories).

# Usage as portscanner

Used like this, `netcat` will test some host for open ports:

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

Possible values for `PARAMETER` are:

Parameter	Meaning
<code>-v</code>	show the closed ports as well
<code>-w &lt;seconds&gt;</code>	timeout for connection setup
<code>-u</code>	udp instead of tcp. Best used together with <code>-w</code>
<code>-r</code>	try the ports in random order
<code>-i &lt;seconds&gt;</code>	delay after each tested port

Examples for a tcp- and an 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
```

## Links to advanced topics

- The [netcat\(1\) manpage](#)
- The [netcat README file](#)