

## NetCat Tutorial

by: Adam Palmer, 06/13/2005

<http://www.securitydocs.com/library/3376>

All information provided here is for educational and development purposes only. Neither LearnSecurityOnline nor me (Adam Palmer) personally will be held responsible for any of your actions either before or after reading this document.

### What is Netcat?

"Netcat is a simple Unix utility which reads and writes data across network connections, using TCP or UDP protocol. It is designed to be a reliable "back-end" tool that can be used directly or easily driven by other programs and scripts. At the same time, it is a feature-rich network debugging and exploration tool, since it can create almost any kind of connection you would need and has several interesting built-in capabilities. Netcat, or "nc" as the actual program is named, should have been supplied long ago as another one of those cryptic but standard Unix tools."

*Taken from the README of the netcat source tree, this description sums up the uses of netcat perfectly.*

Netcat's homepage is: <http://netcat.sourceforge.net>

Throughout this tutorial, I will be giving examples on Linux systems. The official Netcat homepage makes no reference to Windows systems, however I have successfully built Netcat from source under Cygwin, and you can find a Win32 copy built by '@Stake' from: [http://www.atstake.com/research/tools/network\\_utilities/nc11nt.zip](http://www.atstake.com/research/tools/network_utilities/nc11nt.zip) and all examples used below are fully supported under Windows.

Let's examine the netcat syntax before we look at some areas in which netcat can be used:

### Netcat Syntax

```
adam@adamp:~$ nc -h
[v1.10]
connect to somewhere:  nc [-options] hostname port[s] [ports] ...
listen for inbound:   nc -l -p port [-options] [hostname] [port]
options:
    -e prog                program to exec after connect [dangerous!!]
    -b                    allow broadcasts
    -g gateway            source-routing hop point[s], up to 8
    -G num                source-routing pointer: 4, 8, 12, ...
    -h                    this cruft
    -i secs                delay interval for lines sent, ports scanned
    -l                    listen mode, for inbound connects
    -n                    numeric-only IP addresses, no DNS
    -o file                hex dump of traffic
    -p port                local port number
    -r                    randomize local and remote ports
    -q secs                quit after EOF on stdin and delay of secs
    -s addr                local source address
    -t                    answer TELNET negotiation
    -u                    UDP mode
    -v                    verbose [use twice to be more verbose]
```

```
-w secs          timeout for connects and final net reads
-z              zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive]
```

## Netcat Installation

I will cover here three installation methods.

- a. On a debian or similar machine:

```
apt-get install netcat will do the trick:

adamp:~# apt-get install netcat
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
 netcat
0 packages upgraded, 1 newly installed, 0 to remove and 0 not upgrad
Need to get 63.3kB of archives. After unpacking 190kB will be used.
Get:1 http://http.us.debian.org stable/main netcat 1.10-21 [63.3kB]
Fetched 63.3kB in 2s (27.9kB/s)
Selecting previously deselected package netcat.
(Reading database ... 39433 files and directories currently installed)
Unpacking netcat (from ../netcat_1.10-21_i386.deb) ...
Setting up netcat (1.10-21) ...

adamp:~#
```

- b. And for those that prefer RPMs:

```
rpm -Uvh netcat-version.rpm
```

- c. And for those that prefer the source:

```
We will start by wget'ing the source:
adam@adamp:~$ wget http://osdn.dl.sourceforge.net/sourceforge/netcat/netcat-0.7.1.tar.gz
```

We will now untar, cd to the directory we have untarred the source code to, and run the 'configure' script.

```
adam@adamp:~$ tar -xzf netcat-0.7.1.tar.gz
adam@adamp:~$ cd netcat-0.7.1
adam@adamp:~/netcat-0.7.1$ ./configure
```

The configure script should run through with no trouble, as netcat has very few dependencies.

We then run 'make':

```
adam@adamp:~/netcat-0.7.1$ make
```

This will run through and will compile your source, which again should complete simply and successfully. You can then run 'make install' if you have the necessary privileges, or you could simply run 'src/netcat' which will have been built after a

successful 'make'

At this point, you should now have a successful build of netcat somewhere on your system.

## What are the most basic uses?

### Simple File Transfer

So as an example, I will start two copies of netcat on the same machine locally:

```
adam@adamp:~$ netcat -l -p 1111
```

Here, using the `-l` switch, we are able to specify that netcat should go into 'listen mode' i.e. to listen on the specified port. Using `-p 1111` we are able to specify that we are using port 1111. To summarize, netcat will sit and listen for TCP connections on port 1111 and print any data it receives out to the screen.

In another window we start netcat as:

```
adam@adamp:~$ netcat 127.0.0.1 1111
```

This will connect to host 127.0.0.1 (Locally) on port 1111.

We are now able to have a full two way data transmission, in Window 1:

```
adam@adamp:~$ netcat -l -p 1111
This message was typed in WINDOW1
This message was typed in WINDOW2
Now I'm going to end communication with ^C (Ctrl-C)
adam@adamp:~$
```

And in Window 2:

```
adam@adamp:~$ netcat 127.0.0.1 1111
This message was typed in WINDOW1
This message was typed in WINDOW2
Now I'm going to end communication with ^C (Ctrl-C)

adam@adamp:~$
```

This is the most basic use of netcat described. Here, we are using a BASH shell, and thus we may pipe '|' data to and from netcat, as well as using the redirection ('>', '>>', '<', '<<') to allow netcat to integrate into the shell environment. We will now examine using netcat with one of the redirection operators.

Lets say we wanted to simply transmit a plaintext file.

In one window, we will start netcat as:

```
adam@adamp:~$ netcat -l -p 1111 > outputfile
```

This will run netcat with the same parameters specified above, except it will redirect all text received into 'outputfile'.

```
adam@adamp:~$ echo > infile << EOF
> This is a test file.
```

```
> I am going to attempt to transmit this.  
> Using Netcat.  
> EOF  
adam@adamp:~$
```

Here, we have created some text in a file, and this is the file we are going to attempt to transmit:

```
adam@adamp:~$ cat infile | netcat 127.0.0.1 1111 -q 10  
adam@adamp:~$
```

Hopefully this has now been transmitted to the otherside:

```
adam@adamp:~$ cat outputfile  
This is a test file.  
I am going to attempt to transmit this.  
Using Netcat.  
adam@adamp:~$
```

And here we can confirm that it has. The `-q 10` in the command line will quit after EOF (Otherwise netcat will hang waiting for more input for cat and we will have to terminate it manually). The parameter '10' causes it to quit after 10 seconds anyway.

### Tar

Now, there is no reason why we can't integrate tar and netcat together, and use this to transmit a directory across a netcat socket:

```
On one side: tar zcfp - /path/to/directory | nc -w 3 127.0.0.1 1234
```

The tar statement before the pipe tar's and compresses (using gzip) every file within that directory, before printing its output to stdout (The screen). It is then caught by the pipe, and piped to nc which in this example, connects to 127.0.0.1 on port 1234 and sends it the data which would normally hit the screen. The `-w 3` switch causes nc to allow for a 3 second timeout (In the event of a temporary disconnection or similar).

```
On the other side: nc -l -p 1234 | tar xvpz -
```

This will listen on port 1234 for a connection, and will pass any data received to tar. Using the option 'v' we can print out filenames to screen:

```

adam@adamp:~$ ls -alr nc-test/
total 20
-rw-r--r--  1 adam  adam          0 Aug  8 11:57 file1
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir3/
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir2/
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir1/
drwxr-x--x 22 adam  adam       4096 Aug  8 12:01 ../
drwxr-xr-x  5 adam  adam       4096 Aug  8 11:57 ./
adam@adamp:~$ tar zcfp - ~/nc-test | nc -w 3 127.0.0.1 1234
tar: Removing leading '/' from member names
adam@adamp:~$ █

adam@adamp:~$ mkdir nc-test2
adam@adamp:~$ cd nc-test2/
adam@adamp:~/nc-test2$ nc -l -p 1234 | tar xvpz -
home/adam/nc-test/
home/adam/nc-test/dir1/
home/adam/nc-test/dir1/file2
home/adam/nc-test/dir2/
home/adam/nc-test/dir2/file3
home/adam/nc-test/dir3/
home/adam/nc-test/file1
adam@adamp:~/nc-test2$ ls -alr ./home/adam/nc-test/
total 20
-rw-r--r--  1 adam  adam          0 Aug  8 11:57 file1
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir3/
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir2/
drwxr-xr-x  2 adam  adam       4096 Aug  8 11:57 dir1/
drwxr-xr-x  3 adam  adam       4096 Aug  8 12:02 ../
drwxr-xr-x  5 adam  adam       4096 Aug  8 11:57 ./
adam@adamp:~/nc-test2$ █

```

## UDP

Netcat also supports the UDP IP protocol, and this feature can be invoked with the `-u` switch.

### Simple Socket Reply

With what we have learned so far, we are easily able to get netcat to listen in on a socket, and pump out any data we wish when it receives a connection.

As an example:

```
while true; do echo "Leave me alone" | netcat -l -p 1234 -w10; done
```

Consider this line. Firstly lets examine

```
echo "Leave me alone" | netcat -l -p 1234 -w 10
```

What we are doing here, is listening in on port 1234 with a wait time of 10 seconds. If/when we receive a connection, pipe the results of echo "Leave me alone" to netcat. The `-w 10` is necessary, as otherwise any connection made in will remain open forever. We can also optionally add a `-v` in to the netcat command line which will give us verbose information, i.e. who is connecting.

Every time a connection times out (either with the `-w 10` command line switch, or because a connection has been made and then closed), netcat will exit. As this is not what we want, we put the command line within a standard BASH: while CONDITION; do STATEMENT; done clause, which when the condition is set to true will run forever.

### Inetd

If you build netcat with `GAPING_SECURITY_HOLE` defined, you can use it as an "inetd" substitute to test experimental network servers that would otherwise run under "inetd". A script or program will have its input and output hooked to the network the same way, perhaps sans some fancier signal handling. Given that most network services do not bind to a particular local address, whether they are under "inetd" or not, it is possible for netcat avoid the "address already in use" error by binding to a specific address. This lets you [as root, for low ports] place netcat "in the way" of a standard service, since inbound connections are generally sent to such specifically-bound listeners first and fall back to the ones bound to "any". This allows for a one-off experimental simulation of some service, without having to screw around with inetd.conf. Running with `-v` turned on and collecting a connection log from standard error is recommended.

Netcat as well can make an outbound connection and then run a program or script on the originating end, with input and output connected to the same network port. This "inverse inetd" capability could enhance the backup-server concept described above or help facilitate things such as a "network dialback" concept. The possibilities are many and varied here; if such things are intended as security mechanisms, it may be best to modify netcat specifically for the purpose instead of wrapping such functions in scripts.

Speaking of inetd, netcat will function perfectly well \*under\* inetd as a TCP connection redirector for inbound services, like a "plug-gw" without the authentication step. This is very useful for doing stuff like redirecting traffic through your firewall out to other places like web servers and mail hubs, while posing no risk to the firewall machine itself. Put netcat behind inetd and `tcp_wrappers`, perhaps thusly:

```
www stream tcp nowait nobody /etc/tcpd /bin/nc -w 3 realwww 80
```

and you have a simple and effective "application relay" with access control and logging. Note use of the wait time as a "safety" in case realwww isn't reachable or the calling user aborts the connection -- otherwise the relay may hang there forever.

*Inetd/tcp\_wrappers and netcat information, courtesy of: <http://www.spyder-fonix.com/netcat.html>*

### Talking to syslogd -r

Syslog Daemons running with the `-r` switch log not only their own hosts data but accept remote UDP broadcasts. They listen in on UDP port 514.

```
"echo '<0>message' | nc -w 1 -u loggerhost 514"
```

If 'loggerhost' is running syslogd `-r` and can accept your messages. Note the `-u` switch here, to put netcat into UDP mode. Specifying the '`<0>`' before your message ensures that your message receives top priority within syslog (kern.emerg)

### IPv6

We shalln't touch upon IPv6 in this tutorial, as it covers a different area of networking altogether. Syntax and usage is identical though, and the majority of this tutorial will apply. Look out for netcat6 or nc6.

## Internetworking Basics

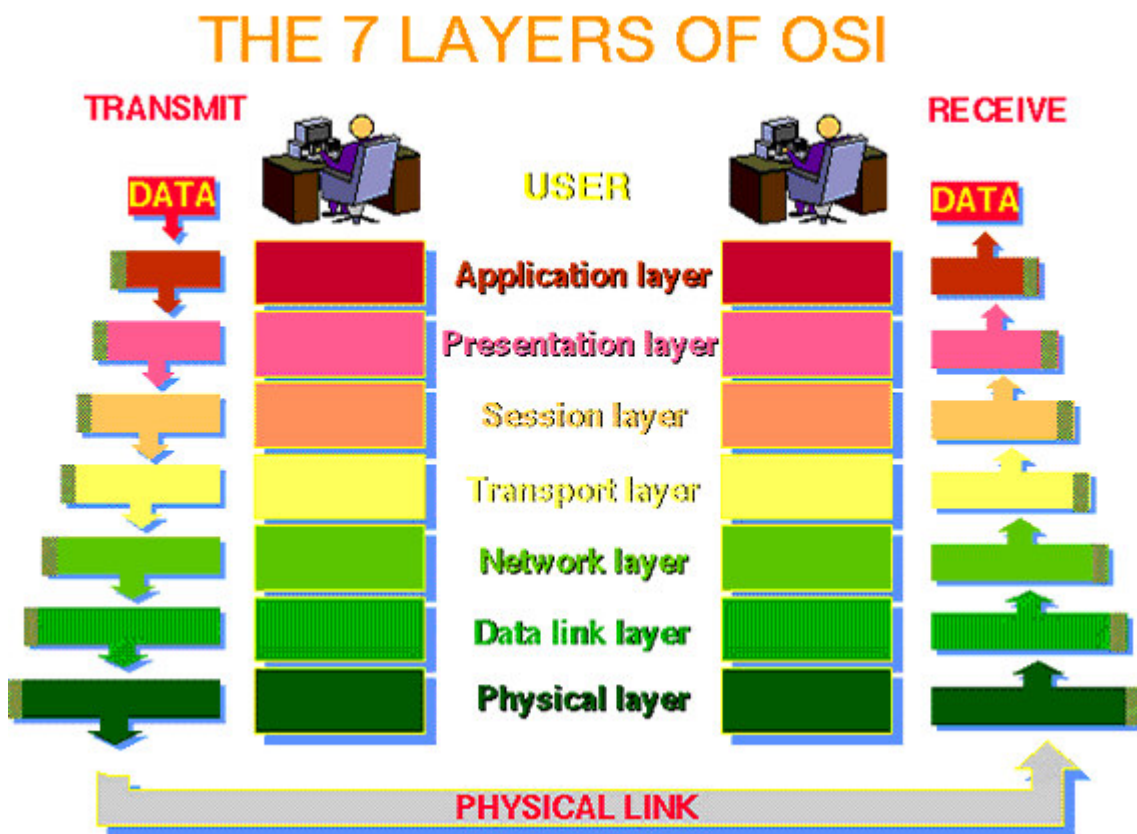
For the purposes of this section, 'machine' refers to an x86 compatible PC with a connection to the Internet through some means, terminated by a standardized TCP/IP stack.

Each machine on the Internet today comes shipped with a standard, compatible TCP/IP stack. This stack guarantees the use of 65535 ports, and IPv4 protocol compatibility.

Below we can see the OSI model. This explains in terms of 7 layers, how data is constructed at one host and received at the next.

In short; Data is constructed on the left by an application, encodes it with a transport (TCP) which takes it over the network (IP), resolves MACs of local devices (Data Link) and then passes a constructed packet to the network card which transmits (Physical) it over the wire (at which point the opposite happens at the other end).

You may have intelligent devices such as switches along the way. These for example may be wise up to layer 5 for example and not only route according to MAC address (Layer 2) but inspect and firewall packets based on findings up to Layer 5 (Simple firewalling) or even Layer 7 (Packet inspection).



"The OSI, or Open System Interconnection, model defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy."

(Courtesy of: [http://webopedia.internet.com/quick\\_ref/OSI\\_Layers.asp](http://webopedia.internet.com/quick_ref/OSI_Layers.asp))

**nc -e**

We have already discussed the basics of redirection with netcat. Netcat has a `-e` switch which we can use to execute a program on connection. There are a couple of viable and legitimate uses for this, i.e. running as `nc -e -v ...` called by the `inetd`

wrapper, which we can use to view traffic and information on users connecting to wrapped daemons, however the most common use which we will explore here is using it to redirect to and from `/bin/bash` or similar shell, for both good and bad.

One method could be this:

```
adam@adamp:~$ nc -v -e '/bin/bash' -l -p 1234 -t
listening on [any] 1234 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 51210
```

In one window, and a simple `telnet localhost 1234` in another window:

```
adam@adamp:~$ telnet 127.0.0.1 1234
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
echo Test
Test
^]
telnet>
```

### Scanning

The scanning features of netcat can be used against yours or your friend's networks to get useful information about which hosts have certain ports open. You can also send a precompiled data file to each. For example:

```
echo EXIT | nc -w 1 127.0.0.1 20-250 500-600 5990-7000
```

Will scan 127.0.0.1 on ports 20-250, 500-600 and 5990-7000. Every port that it finds is open, it will pipe the output of `echo "EXIT"` being the word "EXIT" to that port.

The results are as follows:



```

- PuTTY
adam@adamp:~$ echo EXIT | nc -v -u 1 127.0.0.1 20-250 500-600 5990-7000
localhost [127.0.0.1] 220 (imap3) open
* OK [CAPABILITY IMAP4REV1 X-NETSCAPE LOGIN-REFERRALS AUTH=LOGIN] localhost IMAP4rev1 2001.315 at Sun, 8 Aug 2004 17:58
100 (BST)
EXIT BAD Missing command
localhost [127.0.0.1] 143 (imap2) open
* OK [CAPABILITY IMAP4REV1 X-NETSCAPE LOGIN-REFERRALS AUTH=LOGIN] localhost IMAP4rev1 2001.315 at Sun, 8 Aug 2004 17:58
100 (BST)
EXIT BAD Missing command
localhost [127.0.0.1] 113 (auth) open
0, 0 : ERROR : INVALID-PORT
localhost [127.0.0.1] 80 (www) open
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>501 Method Not Implemented</TITLE>
</HEAD><BODY>
<H1>Method Not Implemented</H1>
EXIT to /index.html not supported.<P>
Invalid method in request EXIT<P>
<HR>
<ADDRESS>Apache/1.3.29 Server at                               Port 80</ADDRESS>
</BODY></HTML>
localhost [127.0.0.1] 53 (domain) open
localhost [127.0.0.1] 25 (smtp) open
220                ZSMTP
500 unrecognized command
localhost [127.0.0.1] 22 (ssh) open
SSH-2.0-OpenSSH_3.4p1 Debian 1:3.4p1-1.uoody.3
Protocol mismatch.
localhost [127.0.0.1] 21 (ftp) open
220 ready, dude (vsFTPD 1.0.0: beat me, break me)
530 Please login with USER and PASS.
localhost [127.0.0.1] 6667 (ircd) open
:chat.              NOTICE * :*** Locking Up Hostname
:chat.              NOTICE * :*** Got Your Hostname
:chat.              451 * :You have not registered
adam@adamp:~$

```

(For the sanity of my server, I have blocked out a number of parts from certain service banners.)

And now with UDP scanning: `nc -v -w 1 127.0.0.1 -u 20-250 500-600 5990-7000` we receive:

```

adam@adamp:~$ nc -u -v -w 1 127.0.0.1 20-250 500-600 5990-7000
localhost [127.0.0.1] 250 (?) open

adam@adamp:~$

```

`-v` was to put netcat into verbose mode, and `-u` was telling netcat to fall into UDP mode.

## Spoofing

"Your TCP spoofing possibilities are mostly limited to destinations you can source-route to while locally bound to your phony address. Many sites block source-routed packets these days for precisely this reason. If your kernel does oddball things when sending source-routed packets, try moving the pointer around with `-G`. You may also have to fiddle with the routing on your own machine before you start receiving packets back. Warning: some machines still send out traffic using the source address of the outbound interface, regardless of your binding, especially in the case of localhost. Check first. If you can open a connection but then get no data back from it, the target host is probably killing the IP options on its end [this is an option inside TCP wrappers and several other packages], which happens after the 3-way handshake is completed. If you send some data and observe the "send-q" side of "netstat" for that connection increasing but never getting sent, that's another symptom. Beware: if Sendmail 8.7.x detects a source-routed SMTP connection, it extracts the hop list and sticks it in the Received: header!" <http://www.spyder-fonix.com/netcat.html>

Spoofing is a useful technique, as is source routing.

Source routing is almost obsolete now, and the majority of routers filter out source routed packets. Source routing in a nutshell is basically setting the route that the packet will take at the source, and storing that information along with the packet. Normally, each router makes its own mind up as to where a packet will get routed, and follows its predefined routing tables. If we have access to all routers between our device and the target device (which can be one machine if you're talking about your local LAN server), then we are able to modify the routing entries on those devices, bind a phoney address to our machine and source route packets to the intended destination.

Spoofing is where we modify the source address of a packet so that the recipient believes it came from a different address. There are two problems with this;

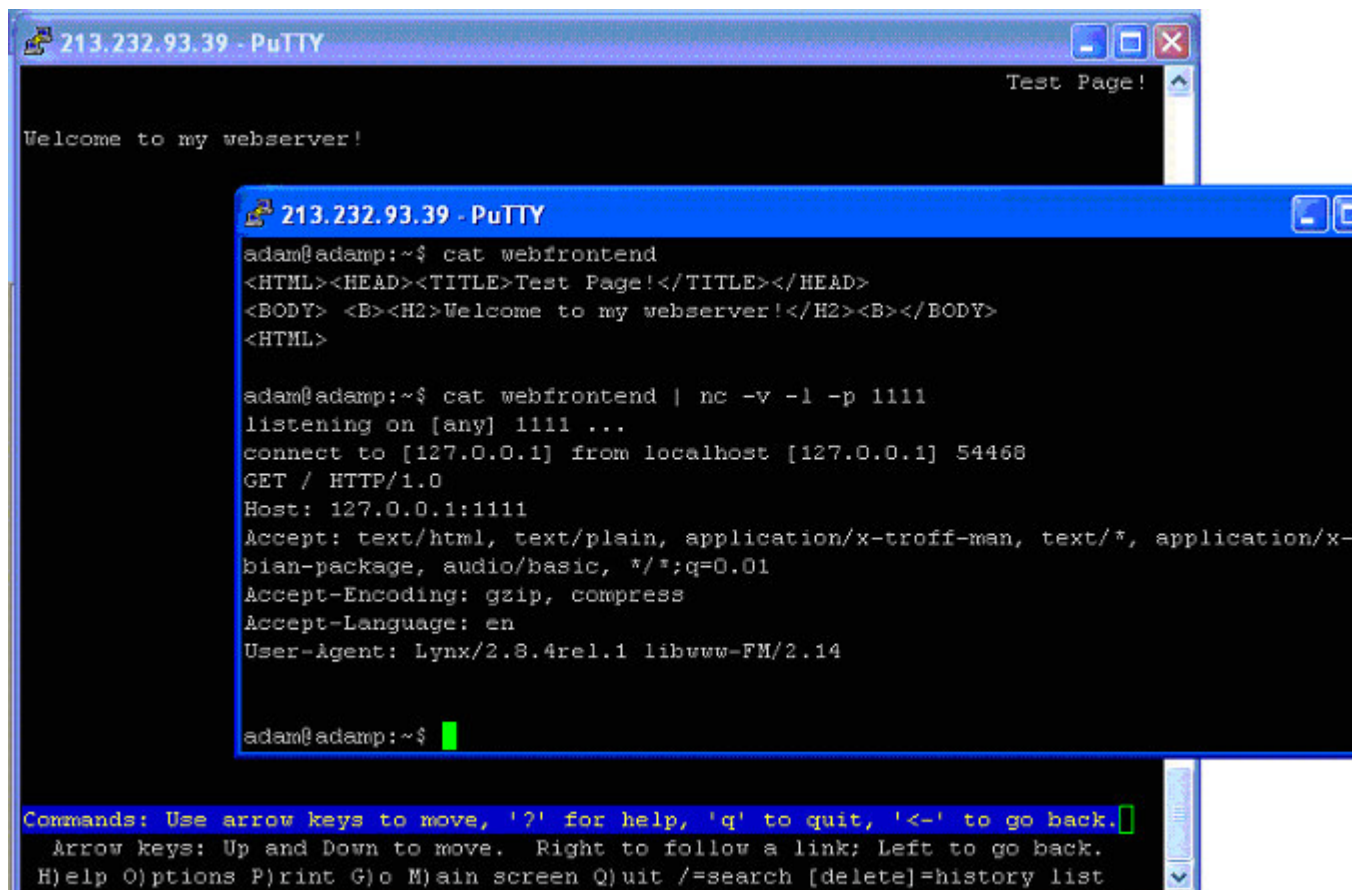
- a. A number of clever ISP routers will drop packets with incorrect source addresses.
- b. If the destination host does get to receive your spoofed packet, it will send data back to the spoofed address (instead of ours). This does have a number of uses however in the example of ICMP ping flooding a host and spoofing the source address to Microsoft.com (as a theoretical example).

### Simple Response Service

```
echo -e "GET http://www.google.com HTTP/1.0nn" | nc -w 5 www.google.com 8
```

We make a connection to google.com on port 80 (Web server port), and put in an HTTP request for http://www.google.com.

At this point, we are presented with the HTML spurted out by the web server. We can pipe this to "| less" or similar or even our favourite HTML interpreter.



The screenshot shows a PuTTY terminal window titled "213.232.93.39 - PuTTY". The terminal displays the output of a web server, including a "Test Page!" header and a "Welcome to my webserver!" message. A second terminal window is overlaid on top, showing the command "cat webfrontend" and its output, which is an HTML document. Below that, the command "cat webfrontend | nc -v -l -p 1111" is shown, along with the output of a netcat listener on port 1111. The netcat listener shows a connection from localhost [127.0.0.1] on port 54468, receiving an HTTP request for "GET / HTTP/1.0". The response from the netcat listener includes the host, accept, accept-encoding, accept-language, and user-agent headers. The terminal also shows a command prompt "adam@adamp:~\$" and a green cursor. At the bottom of the terminal, there is a help message: "Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<->' to go back. Arrow keys: Up and Down to move. Right to follow a link; Left to go back. H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list".

Take a look at this example, and you will see what we have done here. In one instance we have created an HTML file 'webfrontend' and we now pipe that HTML to any incoming connection to netcat on port 1111. We then make a connection on the larger window, using lynx <http://127.0.0.1:1111> and we have made ourselves a tiny http server, possibly could be used as a holding page server or something similar.

### Advanced Proxying

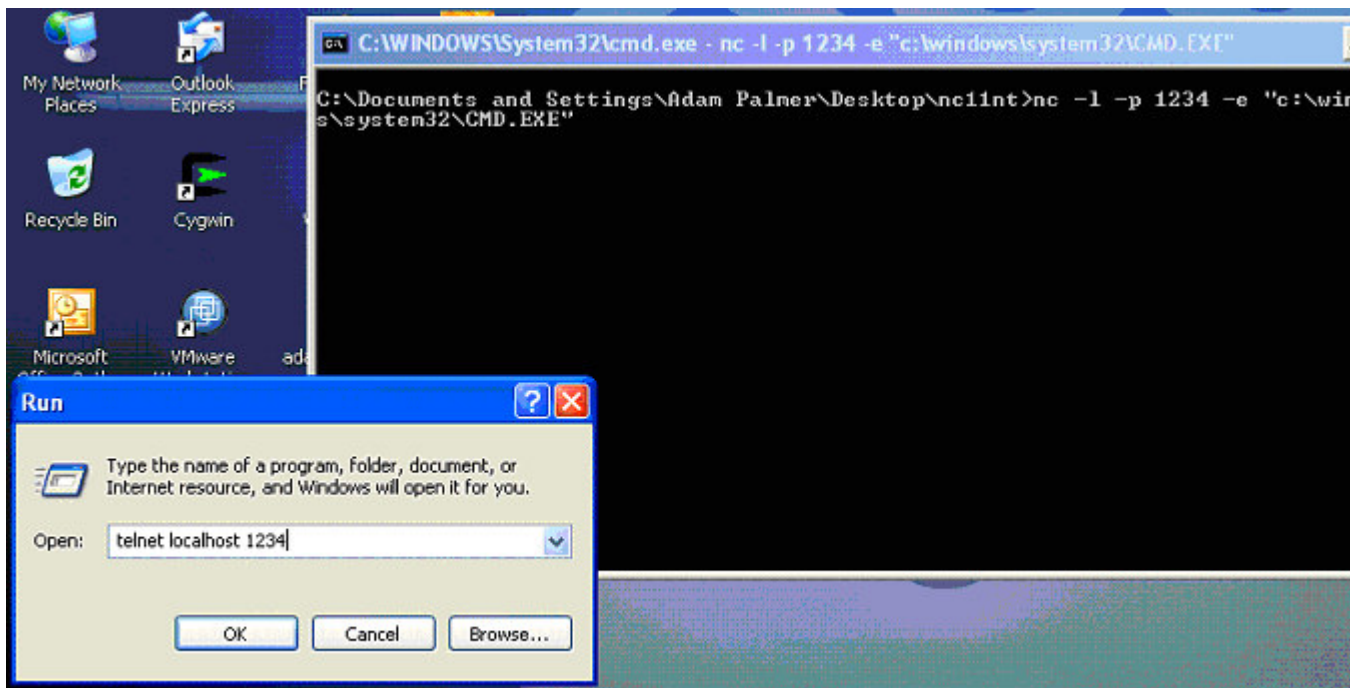
Now we'll set up a server netcat to listen on port 1111. We'll also set up a client netcat to talk to the real web server on port 81. By getting them to pass all data they receive to each other, together they form a proxy; something that sits in the middle of a network connection. Here are the commands we use:

```
mknod backpipe p
nc -l -p 1111 0backpipe
```

Because bash pipes only carry data in one direction, we need to provide a way to carry the responses as well. We can create a pipe on the local filesystem to carry the data in the backwards direction with the mknod command; this only needs to be run once.

Requests coming into the proxy from the client arrive at the first nc, listening on port 1111. They get handed off to the "tee" command, which logs them to the inflow file, then continue on to the second nc command which hands them off to the real web server. When a response comes back from the server, it arrives back at the second nc command, gets logged in the second tee command to the outflow file, and then gets pushed into the backpipe pipe on the local filesystem. Since the first netcat is listening to that pipe, these responses get handed to that first netcat, which then dutifully gives them back to the original client. While the above example is for watching tcp streams going to and from a web server, the above technique is useful for watching any tcp connection. In fact, since nc also works with udp packets - something telnet can't do - it should be possible to even set up udp proxies this way.

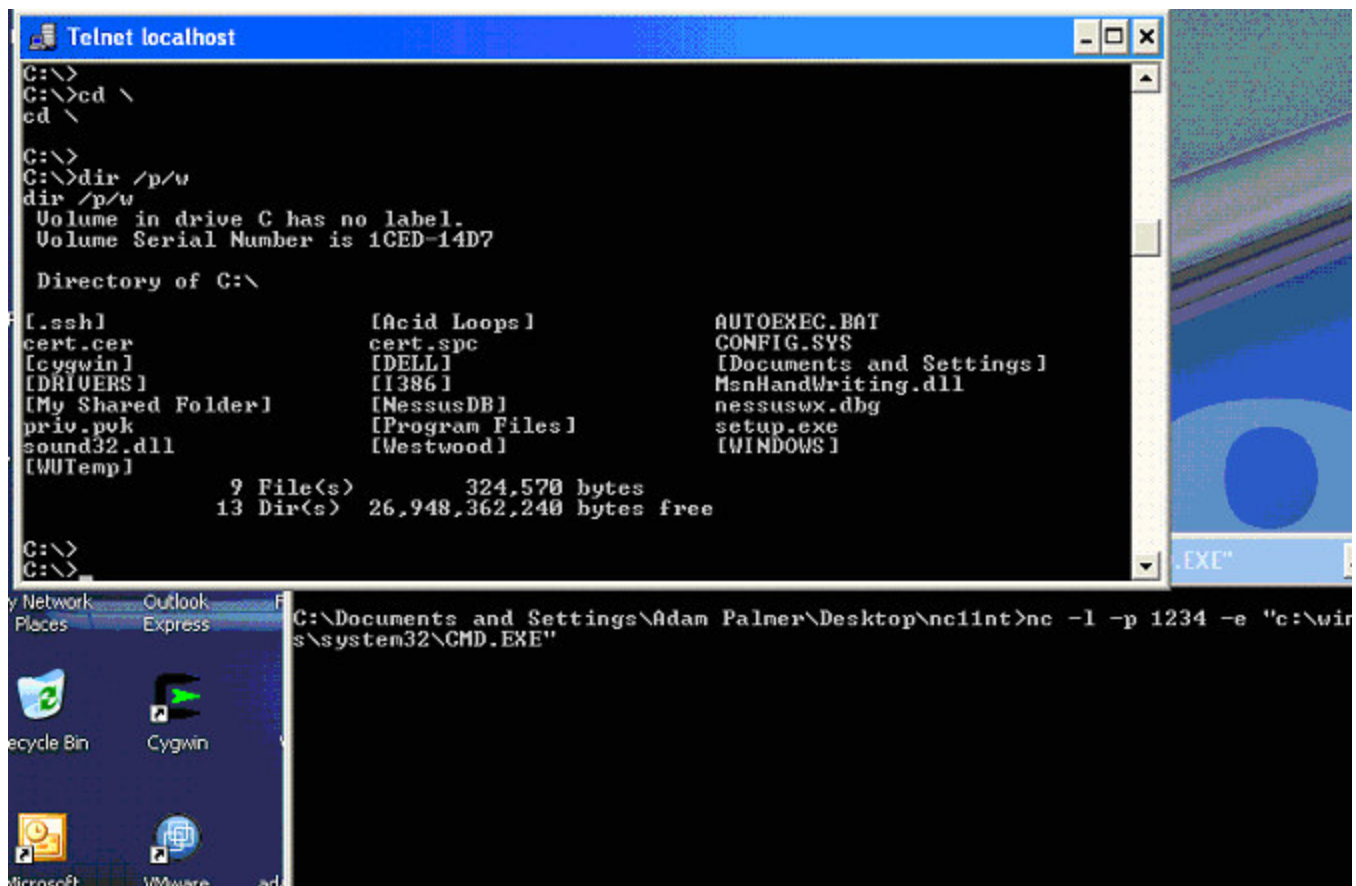
### Windows Command Shell



As we can see from the image above, we have started netcat with options of

```
-l -p 1234 -e "c:\windowssystem32cmd.exe"
```

These are the same options as with the Unix shell, and this should theoretically start a cmd.exe shell listening in on port 1234:



As you see from above, this has succeeded. Netcat and program execution for Windows can be used in exactly the same way.

### Unauthorized Proxying

Assume you're an administrator of a Linux router. Using the methods above, as well as your iptables software, you can proxy a users outgoing connection through your nc proxy. Using iptables with the `-j DNAT` target and the `-j REDIRECT` target, you can transparently proxy outgoing connections through to any other ports you want, and what better to use than your nc proxy?

### Cryptcat

Cryptcat can be found at: <http://sourceforge.net/projects/cryptcat/> and is the ultimate companion for Netcat. It includes a lightweight version of Netcat, featuring encrypted transport properties. (Just for those superbly paranoid!)

### Final Thoughts

If I was given one tool on a freshly installed PC, I would ask for Netcat. Due to its versatility and its huge range of uses, it can be used as a transfer tool, a scanning tool, a server, a proxy and so much more. I have put down everything useful I can think of, and welcome any further suggestions directed to [adam@adamp.co.uk](mailto:adam@adamp.co.uk)

## Command Cheat Sheet

The following are the most useful uses of netcat:

For windows `nc -d` can be used to detach from the console.

<code>nc -l -p [port]</code>	will create a simple listening tcp port. Add <code>-u</code> to put into UDP mode.
<code>nc -e [program]</code>	To redirect stdin/stdout from program.
<code>nc -w [timeout]</code>	To set a timeout before netcat automatically quits. (Used within a loop usually)
<code>program   nc</code>	To pipe output of program to netcat
<code>nc   program</code>	To pipe output of netcat to program
<code>nc -h</code>	Help sheet
<code>nc -v</code>	To put into verbose mode, or use <code>-v -v</code> to put into ultra-verbose mode!
<code>nc -g</code> or <code>nc -G</code>	Source routing flags
<code>nc -t</code>	Use telnet negotiation (If connecting to a telnetd or acting as a telnetd for telnet clients).
<code>nc -o [file]</code>	Hex dump traffic to file
<code>nc -z</code>	No I/O (Used for scanning ports)