A lot of the shell scripting I do involves piping the output of one command into another, such as:

```
$ cat /var/log/messages | awk '{print $4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15}' | sed −e 's/\[[
```

which shows me the system log file after removing the timestamps and [12345] pids and removing duplicates (1000 neatly ordered unique lines is a lot easier to scan than 6000 mixed together). The above technique works well when all the processing can be done on one machine. What happens if I want to somehow send this data to another machine right in the pipe? For example, instead of viewing it with less on this machine, I want to somehow send the output to my laptop so I can view it there. Ideally, the command would look something like:

```
$ cat /var/log/messages | awk '{print $4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15}' | sed −e 's/\[[
```

That exact syntax won't work because the shell thinks it needs to hand off the data to a *program* called laptop − which doesn't exist. There's a way to do it, though.

Enter Netcat

Netcat was written 5 years ago to perform exactly this kind of magic − allowing the user to make network connections between machines without any programming. Let's look at some examples of how it works.

Let's say that I'm having trouble with a web server that's not returning the content I want for some reason. I'd like to see *exactly* what it's sending back for a particular request:

```
echo −e "GET http://mason.stearns.org HTTP/1.0\n\n" | nc mason.stearns.org 80 | less
```

Here's what I see in less:

```
HTTP/1.1 200 OK
Date: Sun, 12 Nov 2000 22:56:42 GMT
Server: Apache/1.3.3 (Unix)  (Red Hat/Linux)
Last−Modified: Thu, 26 Oct 2000 05:13:45 GMT
ETag: "15941−577c−39f7bd89"
Accept−Ranges: bytes
Content−Length: 22396
Connection: close
Content−Type: text/html

<html>
<head>
<title>Mason − the automated firewall builder for Linux</title>
<META NAME="keywords" CONTENT="firewall, Linux, packet, filter, ipfwadm, ipchains, automated, rul
</head>
<body>

<center><img src="mason−banner.gif"></center>
...
```

The echo command created an HTTP request for me. The HTTP protocol requires two linefeeds at the end of the request, so I use two \n's to create them. The actual netcat command (nc) is given this request on stdin. The command line parameters I used ("mason.stearns.org 80") tell netcat to open a connection to port 80 on the server and hand the "GET http..." command to it. Netcat then listens for the reponse (the web page with all the headers) and hands it off to the less command. You can picture netcat working like this:

```
        Web Server
         ^   |
         |   v
         ^   |
         |   v
echo --> netcat --> less
```

The line leading up to "Web" is the outbound web request, and the line leading down from server is the returned web page.

Now I can see the exact html returned from the server, allowing me to troubleshoot what's going on in the html request.

Your own servers − with no programming!

We've just seen how netcat can be used as a tcp/ip client program, allowing us to connect to a remote server, send a request, and pass off the response to some other program. It can also be used as a server.

I'd like to set up a simple server that reports the stable and development versions of the Mason software I write. I'd like it to be available on port 1500 of mason.stearns.org. I'd also like to avoid having to write yet another piece of software to do it. While it's true that I could place this in some finger−accessible file, I'd rather not run the finger daemon as that tends to be a security and privacy risk. First of all, I create a file holding the information I want to make available:

```
<pre>
The latest versions of Mason are:
STABLE: 0.13.0
DEVELOPMENT: 0.13.9.3
</pre>
```

Why the use of the <pre> tags? I want this information to be available to netcat clients and web browsers. If I leave off the <pre> tags, web browsers will just display it all on one line. Ok, now we instruct the server to serve that file up when a client asks for it:

```
while true ; do cat /home/wstearns/mason−version  | nc −l −p 1500 | head −−bytes 2000 >>/tmp/requ
```

The "while true; do....done" runs this command in a constant loop. Since netcat serves up a single file and exits, this immediately starts up netcat again after it has served up a file so it's ready for the next request.

"cat" feeds that version file to netcat; this is the text I want sent back to the user.

"nc −l −p 1500" tells netcat to *l*isten on *p*ort *1500*. As soon as it gets a connection on port 1500, it gives the mason−version text to the remote client. Up to 2000 bytes of any requests the clients send are appended to /tmp/requests along with the timestamp of the request. This provides a simple logging feature, without the risk of someone feeding me gigabytes of data to that port and filling the drive that holds /tmp.

From a client machine, I can now connect to the server to see what the latest versions of Mason are:

```
[wstearns@sparrow wstearns]$ nc localhost 1500
<pre>
The latest versions of Mason are:
STABLE: 0.13.0
DEVELOPMENT: 0.13.9.3
</pre>
```

```
[wstearns@sparrow wstearns]$
```

/tmp/requests contains "Sun Nov 12 18:33:01 EST 2000" as soon as the answer goes back. I can also see this data in a web browser by pointing my web browser to http://mason.stearns.org:1500 . The same text, minus the <pre> tags, shows up in the web browser. You can also use "telnet mason.stearns.org 1500"; it returns the same text as "nc localhost 1500".

The log file shows the headers the web browser sent:

```
GET / HTTP/1.0
User−Agent: Mozilla/4.75 [en] (X11; U; Linux 2.4.0−test11 i686)
Pragma: no−cache
Accept: image/gif, image/x−xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept−Encoding: gzip
Accept−Language: en
Accept−Charset: iso−8859−1,*,utf−8
Via: 1.0 sparrow.websense.net:3128 (Squid/2.3.STABLE4)
X−Forwarded−For: 127.0.0.1
Host: 127.0.0.1:1500
Cache−Control: max−age=259200
Connection: keep−alive

Sun Nov 12 18:39:47 EST 2000
```

There's only one last detail involved in actually making a real server to run. I want it to run during the boot process, and I want it to run as a non−root user. Here's the line to add to /etc/rc.d/rc.local :

```
nohup su nobody −c 'while true ; do cat /home/wstearns/mason−version  | nc −l −p 1500 | head −−by
```

By replacing /home/wstearns/mason−version with the file you want to show, and /tmp/requests with the log file you want to use, you have a ready−to−run server for any kind of data. If you don't want to do any logging, replace ">>/tmp/requests" with ">>/dev/null", and leave off the "; date >>/tmp/requests".

A generic tcp proxy

There's one rather interesting use for netcat that comes in very handy when debugging network traffic. Remember our first example? We wanted to see the exact stream of characters returned from a remote server. What if we want to see all the requests coming in from people to one of our servers, and exactly what is sent back to them? It's not all that hard.

Let's monitor the web server we have on mason.stearns.org. First of all, we need to tell that server to listen on another port, say 81. That's done by editing "/etc/httpd/httpd.conf", changing "Listen 80" to "Listen 127.0.0.1:81" and restarting the web server.

Now we'll set up a server netcat to listen on port 80. 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 80 0<backpipe | tee −a inflow | nc localhost 81 | tee −a outflow 1>backpipe
```

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 80. 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.

The exact form of the nc−tee−nc−tee command line will depend on whether this will be started by hand or in a boot script, and whether you want it to restart automatically or you just need to look at a single connection. Something similar to the above "nohup su nobody −c 'while...done' &will give a persistent proxy startable from the boot scripts, but this may need a little tweaking.

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.

For more ideas, see the examples included in the netcat package. irc and more formal web clients, as well as file copy and simple shell servers are included. Netcat should be available with your distribution, or see contrib.redhat.com for source and architecture specific rpms.

Many thanks to Ed Skoudis and the SANS organization for the one−way proxy idea. If you're interested in host and network security − and I have to assume you are if you're reading my columns − I'd strongly recommend their training. See http://www.sans.org for more info.

William is an Open−Source developer, enthusiast, writer, and advocate from New Hampshire, USA. His day job at SANS pays him to work on network security and Linux projects.

Last updated 12/18/2003.