FUN WITH NETWORK PIPES

This article is about something I find very useful, and very scary at the same time, network pipes. The ability to pipe what you want to where you want it. I'll mostly be using *netcat* (I'm using version 0.7.1) because it is so very good for this, but also because it is small, free and cross platform. I will be touching on ssh briefly, but the concepts are similar.

Tunneling
Yes, I know I've done tunneling before (see here), but you can also do them with netcat. And netcat has a much smaller footprint and is more portable, so using it for tunneling is a useful bit of lore. Lets setup the tunnel..

```
[root@lowsend netcat-0.7.1]# nc -t -L 10.0.0.11:22 -n -v -v -p 8099
Listening on any address 8099
```

The switches used are as follows..

-t	This setups a TCP tunnel
-L	This is for the actual forwarding, the syntax is the server:port. The server is the server which has the port you want to forward
-n	Do not do any host resolving
-v	Be verbose, twice for more
-p	this is the port netcat must listen on for connections

Lets see or tunnel in action..

```
[root@localhost root]# ssh -p 8099 bob@10.0.0.50
bob@10.0.0.50's password:
[bob@syplh bob]$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:10:5A:32:1A:61
          inet addr:10.0.0.11  Bcast:10.0.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:63897 errors:3 dropped:0 overruns:0 frame:5
          TX packets:45977 errors:0 dropped:0 overruns:0 carrier:0
          collisions:254 txqueuelen:100
          RX bytes:16639255 (15.8 Mb)  TX bytes:7038322 (6.7 Mb)
          Interrupt:9 Base address:0xe880

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:7430 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7430 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:576104 (562.6 Kb)  TX bytes:576104 (562.6 Kb)
```

As you can see, while we used ssh to connect to 10.0.0.50 with port 8099, we ended up on the server 10.0.0.11. This is very useful for many varied purposes.

Piping Anything
Another valuable use of netcat is that you can pipe anything you want across the network, files, output, anything. Any way you would normally pipe in a linux command line interface, that is what and how you can pipe in and out of netcat. Take a look..

```
[root@localhost sandbox]# nc -l -t -n -p 1234 > ./file.txt
```

This while create a listening netcat server (the -l switch), and pipe anything we recieve on that port into the sepcified file. So lets try..

```
[root@lowsend sandbox]# cat ./file.txt
1
2
3
12
```

```
23
34
[root@lowsend sandbox]# cat ./file.txt | nc -c -n -t -v -v 192.168.10.80 1234
192.168.10.80 1234 open
Total received bytes: 0
Total sent bytes: 15
[root@lowsend sandbox]#
```

Here we piped the file into netcat, which coonected to the specified server (192.168.10.80), specified port (1234) and which closed the connection when an EOF was found (the -c switch is for this). Lets go back to the first server and see whats in our file..

```
[root@localhost sandbox]# cat ./file.txt
1
2
3
12
23
34
[root@localhost sandbox]#
```

There you go, we have sent the contents of the file across the network through netcat, into a file on a second server. Ah, but I hear you saying "So what? we can do that with ftp, or rcp, or even scp." Very true, but thats an example, remember I said you could pipe anything, take a look at this..

```
[root@lowsend sandbox]# tar -c ./netcat-0.7.1 | nc -t -c -n -v -v -l -p 1234
Listening on any address 1234
```

Here we are create a tar file of the directory netcat-0.7.1, but instead of sending it to a file, we are piping it into a netcat server. Now lets go to the other machine..

```
[root@localhost sandbox]# nc -t 10.0.0.50 1234 > ./nc.tar
[root@localhost sandbox]# tar -tf ./nc.tar
./netcat-0.7.1/
./netcat-0.7.1/m4/
./netcat-0.7.1/m4/lib-link.m4
./netcat-0.7.1/m4/inttypes_h.m4
             |
    <lotsa files>
             |
./netcat-0.7.1/Makefile
./netcat-0.7.1/config.h
./netcat-0.7.1/stamp-h1
```

As you can see, we connected to the listening netcat server and sent the data into a specified tar file. And the file is fine, if you look at the tar file, it is perfectly useable. Now, thats a useful tidbit of information.

Other Network Pipes
I mentioned earlier that I would be touching on ssh, because ssh can also pipe across the network. I'll show you with an example using tar again..

```
[root@lowsend sandbox]# tar -c ./netcat-0.7.1 | gzip -9 | ssh 192.168.10.80 "cat - >
nc.tar.gz"
```

This creates the tar data, sends it through gzip to be compressed and finally through ssh to a file on a different machine. Lets test the file on the other machine to make sure it is fine..

```
[root@localhost root]# gzip -v -t nc.tar.gz
nc.tar.gz:       OK
[root@localhost root]# cat ./nc.tar.gz | gzip -d | tar -t
./netcat-0.7.1/
./netcat-0.7.1/m4/
./netcat-0.7.1/m4/lib-link.m4
./netcat-0.7.1/m4/inttypes_h.m4
./netcat-0.7.1/m4/lcmessage.m4
            |
    <lotsa files>
               |
./netcat-0.7.1/Makefile
./netcat-0.7.1/config.h
./netcat-0.7.1/stamp-h1
```

So you can see, ssh can also be used to pipe data across the network, and in certain cases is preferable as it is encrypted.

Proper Use
Netcat has another handy little use as far as network pipes go, it can listen for a connection and when one is made, it can then execute a file at that time. Now this can be very useful, because if you want to query a process listing, you want the list from when you connected, not from when the netcat server was setup. But -and it is a big but-, this has a dark side. This also means that someone can create a tunneled shell, or can execute any command they want, when they want, but worse yet they can do it on any port they want. Because of the huge potential for abuse most versions of netcat have this switched off, and even if you compile from source you have to explicity turn it on. Lets take look..

```
[root@lowsend netcat-0.7.1]# ./configure CFLAGS=-DGAPING_SECURITY_HOLE && make && make
install

[root@lowsend sandbox]# nc -c -n -t -v -v -l -e "vmstat 3 3" -p 4567
Listening on any address 4567
```

This shows me compiling netcat from source with the feature mentioned (just so that you remember what it is you are doing, they have made sure to name this feature GAPING_SECURITY_HOLE), and once finished, creating a netcat server which will excute the vmstat command on connection.

```
[root@localhost root]# nc -t -n 10.0.0.50 4567
   procs                      memory    swap          io     system         cpu
 r  b  w   swpd   free   buff  cache  si  so    bi    bo   in    cs  us  sy  id
 0  0  0      0  17452  55236  54332   0   0     0     0    2     5   0   0   2
 0  0  0      0  17452  55236  54332   0   0     0     3  111     6   1   2  97
 0  0  0      0  17452  55236  54332   0   0     0     2  109     6   1   1  97
[root@localhost root]#
```

There you have it, when we connected from another system, the netcat server executed the specified command, and when the command finished, the server closed down.

Final Words
Like I said at the start, I have found these little tricks useful for a variety of purposes, but make no mistake, they can be abused. So why show people how to do it? Because the bad guys already know this stuff, now that you know about it too, you know the risks, and can take measures to secure it. As always, have fun and learn.