Emulating Diagnostic Services

Emulating Diagnostic Services

There are a number of simple Internet protocols intended for testing and measurement purposes. Because they deal with simple, fundamental network operations they are a good match for Ncat's capabilities. This section shows how to to emulate services of increasing complexity: discard, echo, daytime, qotd, and chargen. These particular commands assume you are on a UNIX system such as Linux or Mac OS X, and using a `/bin/sh` compatible shell, such as Bash.

The discard service, defined in RFC 863, simply ignores anything sent to it. It runs on TCP or UDP port 9. By default, Ncat doesn't send any information unless instructed to, so nothing special is needed to emulate discard. Send Ncat's output to `/dev/null` to avoid filling the screen with characters received, or just let it write to the terminal if you're curious to see what's there. Use the `--recv-only` option to prohibit sending any characters that might be entered at the terminal.

TCP discard server

```
ncat -l --keep-open 9 --recv-only > /dev/null
```

UDP discard server

```
ncat -l 9 --keep-open --udp --sh-exec "cat > /dev/null"
```

With the TCP server we used `--keep-open` so the server could handle multiple simultaneous connections, not just one. For the UDP server we had to use `--sh-exec` to allow multiple concurrent connections. Recall from the section called "Basic usage" that a UDP server can handle only one client but with `--exec` and `--sh-exec` this limitation does not apply.

The echo service is defined in RFC 862. It runs on TCP or UDP port 7. One step more advanced than discard, it sends back any data received until the connection is closed. How do you instruct Ncat to return what it receives? One easy way is to run everything through `/bin/cat`.

TCP echo server

```
ncat -l 7 --keep-open --exec "/bin/cat"
```

UDP echo server

```
ncat -l 7 --keep-open --udp --exec "/bin/cat"
```

The daytime service, defined in RFC 867, sends a human-readable date and time string over TCP or UDP port 13. It ignores any input. The format of the date and time string is left unspecified, so we are free to use the output of `/bin/date`. Because we are not interested in anything sent by the client we use the `--send-only` option.

TCP daytime server

```
ncat -l 13 --keep-open --send-only --exec "/bin/date"
```

UDP daytime server

```
ncat -l 13 --keep-open --udp --send-only --exec "/bin/date"
```

Nmap comes with a `daytime.nse` script that works with the daytime service. Here is its output running against Ncat daytime servers on TCP and UDP.

Example 10. `daytime.nse` against an Ncat daytime server

```
# nmap -sSU -p 13 --script=daytime localhost
Starting Nmap ( http://nmap.org )

Nmap scan report for localhost (127.0.0.1)
PORT   STATE SERVICE
13/tcp open  daytime
|_daytime: Mon Jan 19 17:43:18 MST 2009
13/udp open  daytime
|_daytime: Mon Jan 19 17:43:18 MST 2009

Nmap done: 1 IP address (1 host up) scanned in 0.31 seconds
```

The qotd (quote of the day) service is defined in RFC 865. When a connection is made to TCP or UDP port 17, it sends back a short message, ignoring any input. Ncat can do this by invoking a program that generates messages. A traditional choice is `/usr/games/fortune`, though there are many possibilities. `/usr/bin/uptime`, for example, could be useful.

TCP qotd server

```
ncat -l 17 --keep-open --send-only --exec "/usr/games/fortune"
```

UDP qotd server

```
do ncat -l 17 --keep-open --udp --send-only --exec "/usr/games/fortune"
```

In this example it's instructive to consider the difference between **ncat -l 17 --exec "/usr/games/fortune"** and **/usr/games/fortune | ncat -l 17**. Think about why the second command stops working after the first connection.

The chargen service from RFC 864 rounds out our tour of diagnostic services. It runs on TCP and UDP port 19. With TCP, chargen ignores any input and sends a never-ending stream of data. Never-ending, that is, until the connection is closed by the user, who the RFC suggests may have “had enough”. There are many ways of generating the characters; reading from `/dev/zero` and running **yes** come to mind.

TCP chargen server

```
yes "chargenchargenchargen" | ncat -l --keep-open 19 --send-only
```

Notice that in this case the program pipes its output into **ncat** rather than being invoked with `--exec`. For chargen either method would work, because the output of **yes** never changes. Using a pipe requires only one process other than **ncat**, but all users connected simultaneously will see the same output stream in synchrony. If the contents must be independent for each stream, then use the `--exec` method, with the understanding that a new process will be started for each connection.

The UDP chargen protocol is a little different. When a datagram is received, it sends back one datagram containing a random number of characters. Implementing this is starting to get away from Ncat, but one way it could be done with the Bash shell is this:

UDP chargen server

```
ncat -l 19 --keep-open --udp --send-only --sh-exec \
  "yes chargenchargenchargen | dd count=1 bs=$(($RANDOM % 512)) 2> /dev/null"
```

Notice the use of `--sh-exec` rather than `--exec` to allow the use of the shell's environment variables and arithmetic evaluation. Standard error is redirected to `/dev/null` to avoid including **dd**'s summary lines (`1+0 records out`), which would otherwise be included by Ncat.

This completes the tour of simple diagnostic services. These have been easy to implement with Ncat because (with the exception of UDP chargen) they all map directly onto a familiar command-line program. As services become more complex it gets harder to do everything in the shell. For complicated services it's better to write a separate program and have Ncat exec it directly.


Neat Tricks

[Nmap | Sec Tools | Mailing Lists | Site News | About/Contact | Advertising | Privacy]

Search