Notes

Using netcat to build a simple TCP proxy in Linux

By Soultaker on Monday 11 June 2012 22:18 - Comments (5)
Category: -, Views: 13.384

Recently I wanted to analyze the network communication between a local application and a remote server. The usual way to accomplish this is to use a system-wide packet analyzer like Wireshark or tcpdump. These tools require the system to be set up to allow packet filtering with special privileges granted to the local user that wants to inspect the network traffic passing through the system.

A less intrusive approach that doesn't require system-wide configuration is the use of a TCP proxy: a program that accepts a local connection, connects to a remote host, and forwards traffic between the two sockets, while also printing out the forwarded data for inspection by the user. Indeed, Google lists many such tools, but Linux users don't need to resort to installing third-party applications when the same functionality can be recreated using already-installed tools!

The tools that we will use are GNU netcat, sed, tee, the ability to create named pipes (also called FIFOs, because they operate like queues with first-in first-out semantics) and the shell to tie everything together.

To create the TCP proxy, we will need two instances of netcat: one to listen for a TCP connection on the local host, and one to connect to the remote host. The output from the first instance should be fed as input into the second instance, and vice versa. We can use an (unnamed) pipe to make one connection, but the shell only supports linear pipelines, not loops. Fortunately, we can use a named pipe to connect the output of the pipeline to its input.

For example, if we want to create a pipeline that proxies an HTTP connection, we could achieve that like this:

```
mkfifo fifo
nc -l -p 8080 <fifo | nc tweakers.net 80 >fifo
```

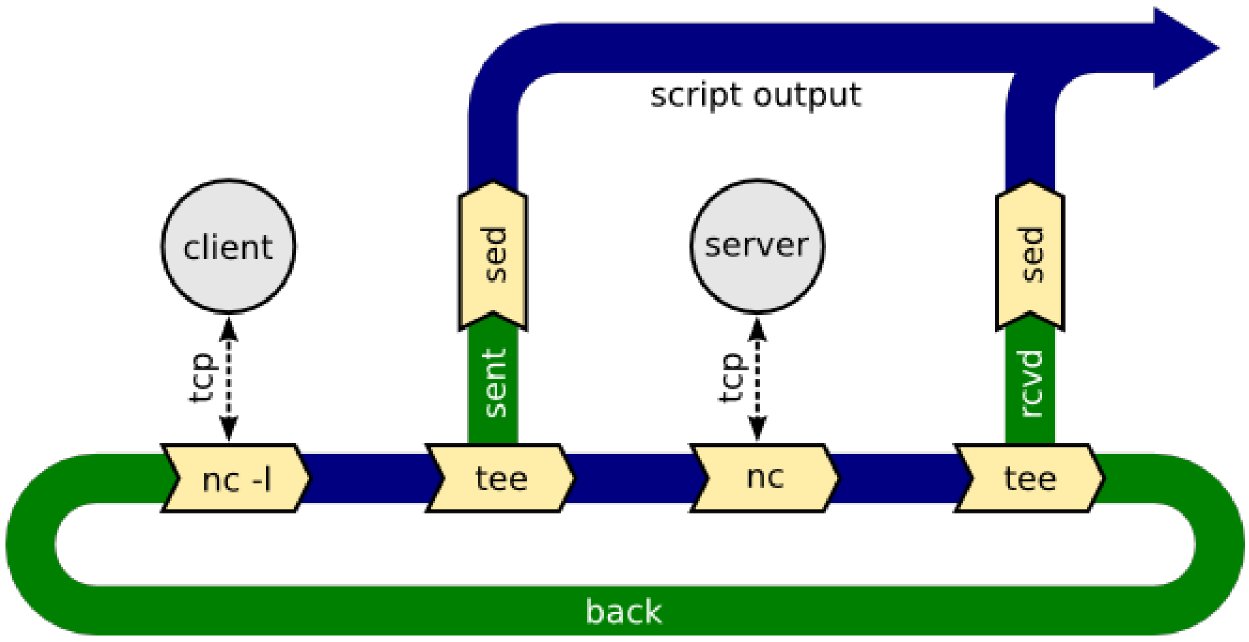
Now we can use a command like the following to send a request through our proxy:

```
http_proxy=localhost:8080 curl -I tweakers.net
```

Of course, at this point, we've only redirected the data, but we aren't able to view it, which was the whole point! To achieve this, we can use the *tee* utility, which forwards its input to its output while also copying it to another file.

Although we could use two tees to write data sent and received to two separate files, that way we lose the correlation between requests and responses. I prefer to merge the output from the two tees, while prefixing each line with a little arrow to indicate whether this line of data was sent by the client or received from the server. We can use two more named pipes to collect this data, and use *sed* to transform it, before writing it to the script's output stream.

This configuration is getting somewhat complicated, so perhaps a visual representation is clearer:



With this design clearly in mind it isn't too hard to write a reusable script to implement it.

shell:

```
#!/bin/sh -e

if [ $# != 3 ]
then
    echo "usage: $0 <src-port> <dst-host> <dst-port>"
    exit 0
fi

TMP=`mktemp -d`
BACK=$TMP/pipe.back
SENT=$TMP/pipe.sent
RCVD=$TMP/pipe.rcvd
trap 'rm -rf "$TMP"' EXIT
mkfifo -m 0600 "$BACK" "$SENT" "$RCVD"
sed 's/^/ => /' <"$SENT" &
sed 's/^/<=  /' <"$RCVD" &
nc -l -p "$1" <"$BACK" | tee "$SENT" | nc "$2" "$3" | tee "$RCVD" >"$BACK"
```

If the script is saved as "tcp-proxy.sh" it can be executed like this:

```
./tcp-proxy.sh 8080 tweakers.net 80
```

Repeating the earlier HTTP request causes the script to print out the request and response data with lines prefixed according to the direction of network traffic:

```
 => HEAD HTTP://tweakers.net HTTP/1.1
 => User-Agent: curl/7.26.0
 => Host: tweakers.net
 => Accept: */*
 => Proxy-Connection: Keep-Alive
 =>
<=  HTTP/1.1 200 OK
<=  Server: Apache
<=  X-Tweakers-Server: phobos
<=  Expires: Mon, 26 Jul 1995 05:00:00 GMT
<=  Last-Modified: Mon, 11 Jun 2012 19:23:09 GMT
[..]
```

Closing Remarks

1. This configuration works best with text-only line-based protocols. Although it's possible to replace the sed processes with e.g. hexdump (which prints binary data in a human-readable format), the output of the two processes gets messed up because the script output is still line-buffered.
2. An advantage of the above script is that it can be easily customized to format traffic in different ways, or even dynamically alter the netwerk conversation, by inserting tools like *grep* or *sed* into the pipeline. For example, we might insert *grep -v ^Proxy-Connection:* to filter out the non-standard HTTP header that curl sends to the webserver.
3. The GNU version of netcat differs slightly from both the original netcat and the FreeBSD port of netcat. Although any implementation could be used in principle, some modifications to the script may be necessary.

Comments

By i-chat, Tuesday 12 June 2012 07:41

in een woord, W0W ... toegegeven, ik moest bij sommige stukjes eff google'n om je weer te volgen... ik zou er nooit opgekomen zijn....

[Comment edited on Tuesday 12 June 2012 07:42]

By afraca, Tuesday 12 June 2012 09:21

Ongeveer the same hier, maar dan met manpages in plaats van google (pff). Vooral de constructie met het mergen van de output streams is leuk 🙂

Dit zijn van die typische dingen waarvan ik enigszins weet dat het kan, maar zelf niet de kennis heb om het aan te pakken. Kan nog wel eens van pas komen, dank.

[Comment edited on Tuesday 12 June 2012 09:22]

By Demoniac, Tuesday 12 June 2012 11:41

Nice blog! Take a look at socat, it's like netcat on steroids I use it myself for quick and dirty audio multicasting (pipe raw PCM to socat on the sending side, pipe it from socat to aplay on the receiving side)

By Soultaker, Tuesday 12 June 2012 20:10

socat looks useful! There's also Ncat (from the developers of Nmap). Both tools seem to offer more advanced options than the traditional netcat (and its clones) but you're less likely to have them installed already.

By Dan, Thursday 09 May 2013 23:24

This script helped me debug a scanner failing to email issue I had.

The only issue I can't get round was it only handled one connection, and needed to be restarted.

Comments are closed