Inleiding Telematica, 2007
0440949 Andreas van Cranenburgh
Sat Apr 21 15:39:22 CEST 2007

Abstract

Though SMTP was never meant to be used for mail submission, this base practice has become all pervasive as of to-day. What follows is a report on an attempt at a simple command-line implementation of such a tool.

1 Implementation of an SMTP submission client

In the old days, SMTP was spoken only by the initiated; well-configured and implemented servers. Mail clients were expected to invoke *sendmail* locally, which takes on responsibility of the mail. However, with the advent of desktop computers, mail needs to be submitted over the network, to a smarthost (instead of just calling it a stupid desktop...), as the desktop itself is not running its own SMTP daemon. This gives rise to some incorrect implementations, for which servers have to be aware.

This article presents and describes the workings of such a creation, written in C. It is tested to compile on OpenBSD and GNU/Linux; also, it has no special requirements.

1.1 Invocation

After uttering prayers, the program can be invoked as:

```
./smtp from-addr to-addr smtp-host
```

Where the three arguments refer to the sender and recipient address, and the mail server to be contacted, respectively. There are no defaults for these arguments, so the code checks that there are exactly three arguments on the command-line. No further sanity checking of the arguments is done, as this is left to the mail server that will be contacted.

1.2 Connecting

First the smtp-host argument is resolved (ie. hostname is reduced to an IP address). Then an IPv4 socket is created (IPv6 shall have to wait), and the connection is made.

1.3 Protocol

According to protocol[1], the server should emit a banner, stating its willingness to engage an SMTP transaction. It could look like this:

```
220 unstable.nl ESMTP Postfix
```

Then we have to reply by saying hello, followed by something which can identify this. Unfortunately the RFC is much too vague on this, they should've just required a proper FQDN so that this can be used as part of anti-spam techniques. This client will do its best to find its own hostname. If that fails or if the hostname is "localhost" (which should be outright rejected by any SMTP server, because it shouldn't ever be talking to itself, and someone else claiming to be you is definitely going to deliver some spam). Note the use of the newer EHLO instead of HELO, as per RFC 2821:

```
EHLO soh
250-unstable.nl
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

[1] See RFC 821, RFC 2821

It shows that this server supports lots of exciting features, none of which we will use. Now the sender and recipient addresses should be sent:

```
MAIL FROM:<a>
250 2.1.0 Ok
RCPT TO:<b>
permanent failure: 504 5.5.2 <b>: Recipient address rejected:
need fully-qualified address
```

And lo, we got rejected! When a permanent or transient failure code is detected, the client exits with EXIT_FAILURE.

1.4 Transmitting the message

Had the addresses been correct and acceptable to our server, we would have entered the DATA stage, where the actual mail will be transmitted:

```
DATA
354 End data with <CR><LF>.<CR><LF>
```

After this, an RFC 2822 formatted message is supposed to follow. This entails a few headers, an empty line, and the message; followed by a single dot as the last line. Naturally, the sender and recipient addresses are added as headers, as well as the current date (which is carefully formatted to conform to the prescribed RFC 2822 format). A subject is not added, and this could very well be considered a bug (but since this is just a command-line tool, and neither specifying it on the command-line nor inside the body of the message is at all convenient, it is just left out in stead).

Now we turn to STDIN, to read the message body. Chunks of 1024 characters are read from STDIN, carefully converting LF to CRLF if necessary, as well as escaping single dots so that the message is not ended prematurely. If a single dot is encountered it is changed to two dots. The possibility of two dots seems to have been overlooked by our esteemed RFC authors.

Finally, after the message has been transmitted, we wait for the blessing:

```
250 2.0.0 Ok: queued as 2859332535
```

And the ordeal ends with:

```
QUIT
221 2.0.0 Bye
```

That's it, our message is in the server's queue and it's not our problem anymore. The connection is closed and our client exits with a succesful exitcode.

1.5 The code

`https://unstable.nl/andreas/ai/it/smtp.tar.gz`