November 18, 2024

Imapcl Documentation

Tomáš Brablec

Contents

1 Introduction

1.1 IMAP4rev1 Protocol

1.1.1 Overview

As defined in [1], the Internet Message Access Protocol (IMAP) allows clients to access and manipulate electronic mail messages stored on a server. The protocol allows for creating, renaming and removing mailboxes, fetching, moving, deleting messages, and other actions. The messages can be fetched whole, or in parts defined by the Internet Message Format (IMF) [2]. The protocol does not allow for sending messages, this is handled by other protocols, such as Simple Mail Transfer Protocol (SMTP).

1.1.2 Commands and Responses

The protocol consists of client sending commands to the server, and the server replying with responses. The server can also send messages unilaterally, but this will not be used in this project. Each client command is prefixed by a unique message tag, and can be followed by required and optional arguments, depending on the command. The server responds with zero or more *untagged responses* (prefixed with *), and a single *tagged response*, prefixed with the tag of the command, to which the response belongs to. Tagged responses always contain a result of either `OK` (success), `NO` (failure) or `BAD` (protocol error).

1.1.3 Mailboxes and Messages

Session is an interval between selecting a mailbox and leaving it. IMAP messages are identified using their *Message sequence numbers*, which can change within a single session. Message sequence numbers are assigned in an ascending sequence from 1, so for example deleting a message shifts the message sequence numbers of all messages above the deleted message one position down.

Additionally, a message in an IMAP mailbox is uniquely identified using its *Unique Identifier* number (UID), combined with the *UID Validity* number. UID numbers can change between sessions, but only when the UID validity number is also changed.

Messages can have flags, some of which have special meaning. For example, the flag `\Seen` indicates whether a messages was previously fetched, and the flag `\Recent` indicates that this is the first session in which the message can be seen.

1.1.4 Connection Procedure

Connection to an IMAP server typically consists of the following:

1. Establishing TCP connection, and optionally a TLS connection
2. Logging in – IMAP supports multiple authentication mechanisms, but we will use the `LOGIN` command
3. Selecting a mailbox using the `SELECT` command
4. Manipulation with messages, for example using the `FETCH` command
5. Logging out using the `LOGOUT` command

1.1.5 Commands

Note: this is not a list of all IMAP commands, only a subset needed for this application

- `LOGIN` – used for authentication using username and password.
- `SELECT` – used for selecting (opening) a mailbox. The server responds with additional information about the opened mailbox, such as the current UID validity value.

- `SEARCH` – lists the sequence numbers of messages specified by the arguments (namely `ALL` for all messages and `UNSEEN` for messages without the flag `\Seen`). The variant `UID SEARCH` returns UIDs instead of sequence numbers.
- `FETCH` – fetches the content of a message based on its sequence number (`UID FETCH` accepts UID instead) and sets the `\Seen` flag on the fetched message. It is possible to specify which parts of the message should be fetched. `BODY[]` will fetch the whole message, `BODY.PEEK[HEADER]` will fetch the message headers. Additionally, it will not add the `\Seen` flag to the message.
- `LOGOUT` – gracefully ends the session and logs out the client from the server. The connection is then closed.

Note: Usage of these commands is demonstrated in Section 1.1.6.

1.1.6 Communication Example

Note: messages are prefixed with ">" for client commands and "<" for server responses, these characters are not present in the protocol

```
> TAG0 LOGIN my_username my_password
< TAG0 OK LOGIN Login successful
> TAG1 SELECT INBOX
< * 63 EXISTS
< * 0 RECENT
< * FLAGS (\Seen \Answered \Flagged \Deleted \Draft \Recent $Forwarded)
< * OK [PERMANENTFLAGS (\Seen \Answered \Flagged \Deleted $Forwarded \*)] Can bee
changed permanently
< * OK [UIDVALIDITY 1] UIDs validity number
< * OK [UIDNEXT 1057] Predicted next UID
< TAG1 OK [READ-WRITE] SELECT completed
> TAG2 UID SEARCH UNSEN
< * SEARCH 702 703 706 707
< TAG2 OK SEARCH completed
> TAG3 UID FETCH 702 BODY.PEEK[HEADER]
< * 688 FETCH (UID 702 BODY.PEEK[HEADER] {1437}
< <header bytes>
< )
< TAG3 OK FETCH completed
> TAG4 LOGOUT
< * BYE logout
< TAG4 OK LOGOUT Completed.
```

1.2 TLS

Transport Level Security (TLS) Protocol [3] is a protocol that establishes encrypted communication between two peers on top of TCP. The communication begins with a handshake, where the peers negotiate the parameters of communication, such as the encryption and key exchange method, perform the key exchange, and begin the encrypted communication.

1.2.1 OpenSSL

OpenSSL is a library that implements the TLS protocol, written in C. To open a TLS connection using the C API, the program must:

1. open a TCP socket
2. initialize the OpenSSL context (`SSL_CTX_new`)
3. turn on server certificate validation (`SSL_CTX_set_verify`)
4. create a new connection within that context (`SSL_new`)
5. add the file descriptor of the TCP socket to the connection (`SSL_set_fd`)

6. perform the TLS handshake (`SSL_connect`)
7. send and receive data over TLS (`SSL_read`, `SSL_write`)

When the client wants to provide a specific certificate for server validation, or a path with such certificates, the function `SSL_CTX_load_verify_locations` can be used to set the file and directory paths.

2 Application Design

The goal is to implement a configurable CLI application which will be able to download messages from an IMAP server.

The application has to support the following features:

- both direct TCP and encrypted TLS connection types
- providing custom TLS certificate file/directory
- authentication using username and password from a file
- selecting a mailbox to download from
- Downloading either all messages or just new messages
- Downloading either complete messages, or just message headers

Given these requirements, we can create the following flow diagram describing the communication with the IMAP server.

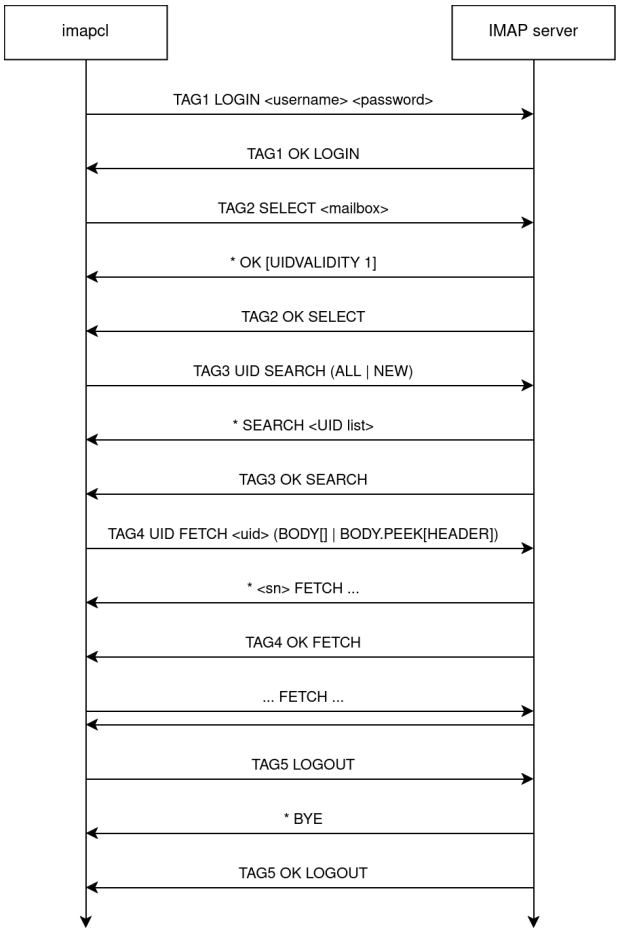

Figure 1: Flow diagram of the communication between `imapcl` and an IMAP server

3 Implementation Details

3.1 Modules

The program consists of the following modules:

- `config.h` - argument parsing and processing, creates a `Config` structure with all program configuration
- `connection.h` - establishes connection (TCP, TLS), creates a `Connection` structure, provides convenient functions to send and receive data
- `imap.h` - implements the IMAP communication itself, sending commands and parsing responses
- `sgc.h`, `utils.h`, `vec.h`, `imports.h` - memory allocator, miscellaneous functions, vector implementation, library imports

3.2 Download Procedure

The application first parses arguments (`config.c`) using `getopt`, then translates server address to IP address using `getaddrinfo`. The credentials are read out of the provided file using `fscanf` into a fixed buffer, so there is an implementation limit of 127 characters for both username and password. The default port is set to 143 for direct TCP connection and to 993 for TLS connection. The resulting `Config` struct is then stored as a global read-only variable to be accessed anywhere in the project. Arguments `-c` and `-C` are ignored if TLS is disabled.

Then the connection is established (`connection.c`). First, the program creates and connects a socket to the server, and then, if TLS support is enabled, also initializes OpenSSL and creates a TLS connection using this socket.

Once the connection is established, `download_messages` is called (`imap.c`). First, the client authenticates using the command `TAG<n> LOGIN <username> <password>` (tag numbers are generated sequentially, which makes every tag unique). Then a tagged response is awaited, and success is checked. The message is sent using a variadic formatting function `imap_write_fmt` from `connection.c`, which will format the command and send it using the correct function (`write` for TCP connection and `SSL_write` for TLS connection).

Then the mailbox is selected using `SELECT <mailbox name>` (from this point, tags are omitted). The IMAP specification places restrictions on the naming of the mailbox, 8-bit names have to be encoded using a custom format. This would be difficult to implement without using an encoding library, and therefore only 7-bit (ASCII) mailbox names are supported. The server sends the UIDVALIDITY number as an untagged response to the `SELECT` command, which is parsed out and stored.

After selecting a mailbox, `UID SEARCH ALL` is used to fetch UIDs of all messages to download. If the `-n` flag was used, `UID SEARCH UNSEEN` is used instead. `UNSEEN` is a shorthand for searching all messages which do not have the `\Seen` flag. This is preferred to using `NEW` because users might want to download new message headers using `-n -h` and then, after checking the subjects, download the whole messages using `-n`. This would not be possible with `NEW`. There is a small problem that some people do not open all of their messages, which would mean that using `-n` would result in downloading many messages. However, this would be a problem only during the first download, which would mark all messages with `\Seen`. After receiving the `SEARCH` untagged response, the UIDs are parsed.

Then, for each UID, a filename is formatted (`get_filename`) with the following structure: `<server_name>-<mailbox_name>-[h-]<uidvalidity>-<uid>`. The `h-` part is present only when

downloading message headers. This filename structure ensures that unless a server or mailbox is renamed, each message will always have its unique file. Then the program checks if such a file already exists. If not, the message is downloaded and stored into this file:

The message is fetched using `UID FETCH <uid> BODY[]`. If the `-h` flag was used, the command `UID FETCH <uid> BODY.PEEK[HEADER]` is used instead. This will fetch only the message header and not mark the message with the `\Seen` flag. This is done because the user will likely use the `-h` and `-n` flags to read the subjects of new messages, and they should still have the option to download the new messages using `-n`. After receiving the untagged `FETCH` response, the message body is read into a buffer, which is then dumped into the file.

After downloading all messages, the program prints the number of newly downloaded messages, sends a `LOGOUT` command to the server, and awaits the tagged response. After that, the `download_messages` function ends, and the program closes the connection, frees all resources and ends.

4 Usage Guide

Usage: `imapcl server [-p port] [-T [-c certfile] [-C certaddr]] [-n] [-h] -a auth_file [-b MAILBOX] -o out_dir`

Required arguments:

- `server` – IP address or hostname of the IMAP server
- `-a <auth_file>` – file containing the username and password for authentication
- `-o <out_dir>` – directory for storing downloaded messages

Optional arguments:

- `-p <port>` – port used for connection (default: 143 for TCP connection, 993 for TLS connection)
- `-T` – enable TLS connection
- `-c <certfile>` – certificate file used for validation of server certificate
- `-C <certaddr>` – directory with certificates for validation of server certificate (default: `/etc/ssl/certs`)
- `-n` – download only new messages
- `-h` – download only message headers
- `-b <mailbox>` – mailbox name to download messages from (default: `INBOX`)

4.1 Examples

4.1.1 All Messages

Download all messages from INBOX on server imap.example.org:

```
./imapcl imap.example.org -a cred -o out
```

The file `cred` should contain the following (newlines at the end):

```
username = my_username
password = my_password
```

4.1.2 New message headers

Download headers of new messages from INBOX on server imap.example.org:

```
./imapcl imap.example.org -a cred -o out -n -h
```

4.1.3 TLS certificate

Download messages from Important on server imap.example.org over TLS using the certificate `cert.pem`:

```
./imapcl imap.example.org -a cred -o out -T -c cert.pem -b Important
```

5 Testing

All tests were executed on `merlin.fit.vutbr.cz` with OpenSSL 3.0.15.

5.1 Note on Valgrind

All of the tests below were retested under Valgrind to discover any potential use of uninitialized or invalid memory. However, when testing any functionality under Valgrind, the following results are reported:

```
valgrind --leak-check=full --show-leak-kinds=all ./imapcl imap.stud.fit.vutbr.cz -a
cred -o out

==23866== 19 bytes in 1 blocks are still reachable in loss record 1 of 9
==23866==    at 0x4843866: malloc (vg_replace_malloc.c:446)
                           ...
==23866==    by 0x400C903: _dl_open (dl-open.c:905)

...

LEAK SUMMARY:
   definitely lost: 0 bytes in 0 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 0 bytes in 0 blocks
   still reachable: 5,391 bytes in 12 blocks
        suppressed: 64 bytes in 2 blocks
```

This is known bug (documented in [4]) and is not caused by this application. This application uses its own allocator wrapper on top of standard `malloc`, which can deallocate all memory at the end of the program. Besides this, testing with Valgrind did not find any errors related to memory use. This is to be expected, because the whole development was done with AddressSanitizer enabled.

5.2 Downloading Messages

5.2.1 First download

- setup – Directory `out` is empty. File `cred` contains correct login credentials.
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out`
- result – `Downloaded 1417 messages from INBOX`

Directory `out` contains 1417 files. Files contain full messages in IMF format.

5.2.2 Subsequent download

- setup – directory `out` contains messages from Section 5.2.1, otherwise the same
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out`
- result – `Downloaded 0 messages from INBOX`

Directory `out` still contains 1417 original files.

5.2.3 New message

- setup – Same as in Section 5.2.2, one new message was sent to the address
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out`
- result – `Downloaded 1 messages from INBOX`

Directory `out` contains new file `eva.fit.vutbr.cz-INBOX-1660639321-1423.eml` with the correct content.

5.2.4 Switch `-n`

- setup – Same as in Section 5.2.1, one new message was sent to the address
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -n`
- result – `Downloaded 1 messages from INBOX`

Directory `out` contains one file `eva.fit.vutbr.cz-INBOX-1660639321-1424.eml` with the correct content.

5.2.5 Switch `-h`

- setup – Same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -h`
- result – `Downloaded 1419 headers from INBOX`

Directory `out` contains 1419 files. Files contain message headers in IMF format.

5.2.6 Switch `-n` and `-h` combination

- setup – Same as in Section 5.2.1, one new message was sent to the address
- commands
  - `./imapcl eva.fit.vutbr.cz -a cred -o out -n -h`
  - `./imapcl eva.fit.vutbr.cz -a cred -o out -n`
- result
  - `Downloaded 1 headers from INBOX`
  - `Downloaded 1 messages from INBOX`

Directory `out` contains 2 files with the correct content:

- `eva.fit.vutbr.cz-INBOX-h-1660639321-1425.eml`
- `eva.fit.vutbr.cz-INBOX-1660639321-1425.eml`

5.2.7 Switch `-b`

- setup – Same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -b Sent`
- result – `Downloaded 28 messages from Sent`

Directory `out` contains 28 files with messages sent from my account.

5.2.8 TLS connection

- setup – Same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -T`
- result – `Downloaded 1420 messages from INBOX`

Directory `out` contains 1420 files. Files contain full messages in IMF format.

5.2.9 TLS connection (custom certificate)

- setup

  - created custom certificate using `openssl req -x509 -newkey rsa -keyout key.pem -out cert.pem -nodes`
  - started TLS server using `openssl s_server -accept 1234 -key key.pem -cert cert.pem`
- command – `./imapcl localhost -p 1234 -a cred -o out -T -c cert.pem`
- result – `imapcl` established connection, OpenSSL printed out `TAG0 LOGIN ...` sent by the client

5.2.10 TLS connection (custom certificate path)

- setup – same as Section 5.2.9
  - rehashed certificates in current directory using `c_rehash .`
- command – `./imapcl localhost -p 1234 -a cred -o out -T -C .`
- result – `imapcl` established connection, OpenSSL printed out `TAG0 LOGIN ...` sent by the client

5.3 Error Handling

5.3.1 Invalid address

- setup – same as in Section 5.2.1
- command – `./imapcl  -a empty -o out`
- result – `Could not connect to server: Connection refused`

5.3.2 Wrong credentials

- setup – same as in Section 5.2.1, file `empty` is empty
- command – `./imapcl eva.fit.vutbr.cz -a empty -o out`
- result – `Could not read auth file 'empty'`

5.3.3 Wrong credentials (2)

- setup – same as in Section 5.2.1, file `empty` contains wrong credentials in correct file format
- command – `./imapcl eva.fit.vutbr.cz -a empty -o out`
- result – `Could not log in`

5.3.4 Wrong mailbox name

- setup – same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -b kentus`
- result – `Could not select mailbox`

5.3.5 Read-only output directory

- setup – same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o /`
- result – `Failed to open file '//eva.fit.vutbr.cz-INBOX-1660639321-1.eml': Permission denied`

5.3.6 Invalid TLS certificate

- setup – same as in Section 5.2.1
- command – `./imapcl eva.fit.vutbr.cz -a cred -o out -T -c cert.pem`
- result – `Could not set specified TLS certificate file/path`

5.3.7 Invalid TLS certificate (2)

- setup – same as in Section 5.2.9
- command – `./imapcl localhost -p 1234 -a cred -o out -T`
- result – `Could not establish TLS connection`

OpenSSL could not verify the self-signed certificate using the system certificates.

5.3.8 Invalid server response

- setup – same as in Section 5.2.8, `-crlf` is added to OpenSSL server
- command – `./imapcl localhost -p 1234 -a cred -o out`
  - `some invalid text` is typed into OpenSSL server
  - no client reaction (client waits for tagged response)
  - `TAG0` is typed into OpenSSL server
  - client: `imap.c:82: Server returned invalid data`

Bibliography

[1] M. Crispin, "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1." [Online]. Available: https://www.rfc-editor.org/info/rfc3501

[2] P. Resnick, "Internet Message Format." [Online]. Available: https://www.rfc-editor.org/info/rfc2822

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3." [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[4] A. Diaz, "leak on fedora 33 and higher in getaddrinfo." [Online]. Available: https://sourceware.org/bugzilla/show_bug.cgi?id=28808