VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Síťové aplikace a správa sítí – Project

IMAP client with TLS support

November 16, 2024

Matúš Ďurica

Contents

Introduction

This project aimed to create a terminal-based IMAP client with TLS support. After execution, the client fetches messages stored on the server and stores them locally one by one. Its output is a message which tells the user how many messages were fetched and stored. The user can tweak the functionality further using command-line arguments.

1 Theory

1.1 Internet Message Access Protocol

The Internet Message Access Protocol, Version 4rev1 (IMAP4rev1) allows a client to access and manipulate electronic mail messages on a server. IMAP4rev1 permits manipulation of mailboxes (remote message folders) in a way that is functionally equivalent to local folders. IMAP4rev1 also provides the capability for an offline client to resynchronize with the server [9].

1.2 Internet Message Format

At the most basic level, a message is a series of characters. A message that is conformant with this specification is composed of characters with values in the range of 1 through 127 and interpreted as US-ASCII [ANSI.X3-4.1986] characters.

Messages are divided into lines of characters. A line is a series of characters that is delimited with the two characters carriage-return and line-feed; that is, the carriage return (CR) character (ASCII value 13) followed immediately by the line feed (LF) character (ASCII value 10).

A message consists of header fields (collectively called ”the header section of the message”) followed, optionally, by a body. The header section is a sequence of lines of characters with special syntax as defined in this specification. The body is simply a sequence of characters that follows the header section and is separated from the header section by an empty line (i.e., a line with nothing preceding the CRLF) [15].

1.3 Transmission Control Protocol

TCP provides a reliable, in-order, byte-stream service to applications. The application byte-stream is conveyed over the network via TCP segments, with each TCP segment sent as an Internet Protocol (IP) datagram. TCP reliability consists of detecting packet losses (via sequence numbers) and errors (via per-segment checksums), as well as correction via retransmission [10].

1.4 Secure Sockets Layer and Transport Layer Security

The primary goal of TLS is to provide a secure channel between two communicating peers; the only requirement from the underlying transport is a reliable, in-order data stream. Specifically, the secure channel should provide the following properties:

- Authentication: The server side of the channel is always authenticated; the client side is optionally authenticated. Authentication can happen via asymmetric cryptography (e.g., RSA, the Elliptic Curve Digital Signature Algorithm (ECDSA), or the Edwards-Curve Digital Signature Algorithm (EdDSA)) or a symmetric pre-shared key (PSK).
- Confidentiality: Data sent over the channel after establishment is only visible to the endpoints. TLS does not hide the length of the data it transmits, though endpoints are able to pad TLS records in order to obscure lengths and improve protection against traffic analysis techniques.
- Integrity: Data sent over the channel after establishment cannot be modified by attackers without detection. [14]

2 Implementation details

2.1 Return Codes

The application provides these return codes to narrow down errors during execution.

0	–	General success
1	–	General failure
2	–	Missing server argument
3	–	Missing required argument
4	–	Missing option for an argument
5	–	Unknown argument
6	–	Missing -T argument when -c/-C/both present
7	–	Socket creating failed
8	–	Connecting to the socket failed
9	–	Bad host
10	–	Socket timed out
11	–	Error while opening authentication file
12	–	Authentication file does not exist
13	–	Invalid credentials
14	–	Missing credentials
15	–	Output directory does not exist
16	–	Error while opening validity file
17	–	Can not access mailbox
18	–	Writing to a socket failed
19	–	Invalid response from the server
20	–	Certificate file or directory error
21	–	Error while creating the SSL context
22	–	Error while creating the SSL connection
23	–	Error while setting the socket descriptor to the SSL
24	–	Error while performing the SSL handshake

2.2 Argument parsing

CLI argument parsing is done using the `getopt` function [11] provided by the `getopt.h` header file.

This function is utilized by the `Utils::CheckArguments` method, which checks all arguments received from the user. When the user inputs invalid or unknown arguments, an error message will be printed, and the application will exit with a relevant return code.

2.3 Sessions

2.3.1 Session

Class `Session` contains an interface for plain text communication with the IMAP server.

It utilizes the functionality of a BSD socket for communication with the IMAP server. Reading and writing to the socket are done by the `recv` [12] and `send` [13] functions, respectively. The communication itself is in plain text; therefore, it is insecure.

It is also a base class for the `EncryptedSession` class.

2.3.2 Encrypted Session

Class `EncryptedSession` contains an interface used for encrypted communication with the IMAP server.

It utilizes functions from the `openssl.h` header file to encrypt a BSD socket, which is then used to send or read encrypted data using the `SSL_write` [8] and `SSL_read` [6] functions, respectively.

Encrypting the BSD socket consists of creating a secure SSL context using `SSL_CTX_new` function, which takes a `method` parameter provided by the `TLS_client_method` function [3]. This is a general-purpose version-flexible SSL/TLS method. The actual protocol version used will be negotiated to the highest version mutually supported by the client and the server [4]. After creating the secure SSL context, the certificate directory (and the certificate itself, if the user supplied the path to it) is specified to the context using `SSL_CTX_load_verify_dir` (`SSL_CTX_load_verify_file`) [2]. Then, the SSL structure that holds the data for an SSL/TLS connection is created using the `SSL_new` function [5]. Then, setting the BSD socket file descriptor to the SSL context is done using `SSL_set_fd` function [7], and finally, the socket is ready to be connected using `SSL_connect` function [1].

2.4 Messages

2.4.1 Message

Class `Message` defines an object for the email message. It is a base class to the `HeaderMessage` class. The file name follows this convention:

`<UID>_<Mailbox>_<Hostname>_<Subject>_<Sender>_<Hash>.eml`

UID	–	UID of the message
Mailbox	–	Mailbox, where the message is located
Hostname	–	Hostname of the IMAP server
Subject	–	Subject of the message
Sender	–	Sender message address
Hash	–	Hashed Message-ID of the message

The message body is extracted from the server response using the `std::string::substr` method - as a first thing, the `RFC822.SIZE` is fetched from the server, representing the number of octets in the message. When parsing the message body, the first thing done is to apply a regular expression to remove the first line of the response, which contains the beginning of the IMAP response, which is not a part of the message body. Then the `RFC822.SIZE` is passed to the `substr` method to get the number of octets of the message. This substring is then stored as the message body.

It also provides methods for setting the file name and content and saving it to a directory.

2.4.2 Header Message

Class `HeaderMessage` defines an object of a message containing only the headers of an email message.

The file name follows the same convention as above, with the only difference being adding `_h` after the hashed Message-ID:
`<UID>_<Mailbox>_<Hostname>_<Subject>_<Sender>_<Hash>_h.eml`

Same as for the `Message` class, the headers are extracted from the IMAP response using a regular expression, which removes the first line of the IMAP response, and making a substring based on the number of the octets of the headers. The only difference is that the number of the octets is extracted from the fetch response using another regular expression, as the RFC3501 protocol does not provide a way to obtain a number specifying the size of the headers.

2.5 Differences to the assignment

My implementation allows the user to build the application with debug information being printed. Please refer to chapter 3.1.3. Since the assignment did not specify a reading timeout, my implementation waits for 10 seconds.

3 Running the Application

3.1 Compilation

3.1.1 Prerequsites

- GNU make
- g++
- netinet/*
- sys/*
- arpa/*
- openssl/*
- libcrypto

3.1.2 Compilation of the application

The IMAP client can be compiled using the provided makefile by using `make` or `make all` from the project's root directory.

3.1.3 Compilation with debugging information

The implementation provides debug mode, in which the user can see more information during the execution of the application. Application with the debug mode turned on can be built using `make debug` from the project's root directory.

3.2 Running the Application

The application can be run using the following command:

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

Where:

- server - Required IP address/hostname of an IMAP server
- -p port - Optional port number

  DEFAULT VALUE:

  143 for unencrypted communication

  993 for encrypted communication
- -T - Turns on encrypted communication (imaps)
- -c certfile - Optional certificate file used for verifying SSL/TLS certificate from the server

- -C certaddr - Optional certificate directory, where certificates for verifying the SSL/TLS certificate from the server will be looked up
- -n - Only new messages will be fetched
- -h - Only headers will be fetched
- -a auth_file - Required path to a file containing username and password for authentication on the server
- -b MAILBOX - Optional mailbox name

  DEFAULT VALUE:

  INBOX
- -o out_dir - Required path to a directory to which messages will be fetched

3.2.1 Authentication File

The authentication file in the following format stores the username and password, which will be used by the application:

```
username = <username>
password = <password>
```

4 Testing

Automated unit tests for testing the parsing of arguments and the authentication file are provided with the project. Integration testing was done manually with `zoznam.sk` dummy account and with my faculty email account.

4.1 Compilation of automated tests

4.1.1 Prerequsites

Automated tests require the same prerequisites as the IMAP client itself, with the addition of:

- Google Test

4.1.2 Compilation of automated tests

Automated tests can be compiled and run using `make test` command from the project's root directory.

4.2 Tests output

Listing 1: Output of automated unit tests

```
[==========] Running 22 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 6 tests from Arguments
[ RUN      ] Arguments.MissingServer
[2] ERROR: Missing server address
[       OK ] Arguments.MissingServer (0 ms)
[ RUN      ] Arguments.MissingAuthFile
[3] ERROR: Missing required argument
[       OK ] Arguments.MissingAuthFile (0 ms)
```

```
[ RUN      ] Arguments.MissingOutDirectory
[3] ERROR: Missing required argument
[       OK ] Arguments.MissingOutDirectory (0 ms)
[ RUN      ] Arguments.BadAuthFile
[12] ERROR: Auth file does not exist
[       OK ] Arguments.BadAuthFile (0 ms)
[ RUN      ] Arguments.BadOutputDirectory
[15] ERROR: Output directory does not exist
[       OK ] Arguments.BadOutputDirectory (0 ms)
[ RUN      ] Arguments.UnknownArgument
[5] ERROR: Unknown argument
[       OK ] Arguments.UnknownArgument (0 ms)
[----------] 6 tests from Arguments (0 ms total)

[----------] 12 tests from ArgumentsMissingOptions
[ RUN      ] ArgumentsMissingOptions.MissingOptionAuthFile
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionAuthFile (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionInbox
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionInbox (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionCertFile
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionCertFile (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionCertDir
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionCertDir (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionOutDir
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionOutDir (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionPort
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionPort (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionAuthFileEnd
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionAuthFileEnd (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionInboxEnd
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionInboxEnd (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionCertFileEnd
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionCertFileEnd (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionCertDirEnd
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionCertDirEnd (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionOutDirEnd
[4] ERROR: Missing required argument option
[       OK ] ArgumentsMissingOptions.MissingOptionOutDirEnd (0 ms)
[ RUN      ] ArgumentsMissingOptions.MissingOptionPortEnd
[4] ERROR: Missing required argument option
```

```
[       OK ] ArgumentsMissingOptions.MissingOptionPortEnd (0 ms)
[----------] 12 tests from ArgumentsMissingOptions (0 ms total)

[----------] 4 tests from AuthenticationFile
[ RUN      ] AuthenticationFile.MissingUsername
[14] ERROR: Missing username or invalid username format
[       OK ] AuthenticationFile.MissingUsername (0 ms)
[ RUN      ] AuthenticationFile.SpacesInUsername
[       OK ] AuthenticationFile.SpacesInUsername (0 ms)
[ RUN      ] AuthenticationFile.MissingPassword
[14] ERROR: Missing password or invalid password format
[       OK ] AuthenticationFile.MissingPassword (0 ms)
[ RUN      ] AuthenticationFile.SpacesInPassword
[       OK ] AuthenticationFile.SpacesInPassword (0 ms)
[----------] 4 tests from AuthenticationFile (2 ms total)

[----------] Global test environment tear-down
[==========] 22 tests from 3 test suites ran. (2 ms total)
[  PASSED  ] 22 tests.
```

5 Limitations of the implementation

During the testing on the faculty email with the `-n` argument, I discovered that the faculty's IMAP server `imap.stud.fit.vutbr.cz` was not searching `NEW` (messages that have the \Recent flag set but not the \Seen flag [9]) messages.

I confirmed this by connecting to the faculty IMAP server using the `openssl` command-line tool and searching for `NEW` messages manually (of course, with newly received messages that could not have had their \Recent flag removed by a previous session). Cross-checking with my dummy email on `zoznam.sk` resulted in the application working as expected.

Therefore, I believe this has to do something with the faculty IMAP server configuration because no other instances of any IMAP clients (i.e. Thunderbird, Outlook) were running on my computer in the background. I also tried running the application on the faculty's `merlin.fit.vutbr.cz` server to no avail.

Because searching for `NEW` messages worked on my dummy `zoznam.sk` account, I decided to keep searching for `NEW` messages instead of searching for `UNSEEN` messages.

Due to no closer specification of the username and password format, the implementation allows spaces between strings. This means that `username = example example` would be evaluated as string `example example`, which in itself is not incorrect, but when authenticating on the IMAP server, this would result in too many arguments in the `LOGIN` command, which would be interpreted as incorrect by the IMAP server, resulting in the application reporting invalid response from the server, rather than the authentication failed due to incorrect credentials, which might be confusing for the user.

Bibliography

[1] AUTHORS, T. O. P. *SSL_connect* [online]. Available at: https://docs.openssl.org/1.0.2/man3/SSL_connect/.

[2] AUTHORS, T. O. P. *SSL_CTX_load_verify_locations* [online]. Available at: https://docs.openssl.org/3.0/man3/SSL_CTX_load_verify_locations/#synopsis.

[3] AUTHORS, T. O. P. *SSL_CTX_new* [online]. Available at: https://docs.openssl.org/master/man3/SSL_CTX_new.

[4] AUTHORS, T. O. P. *SSL_CTX_new* [online]. Available at: https://docs.openssl.org/master/man3/SSL_CTX_new/#notes.

[5] AUTHORS, T. O. P. *SSL_new* [online]. Available at: https://docs.openssl.org/master/man3/SSL_new/#description.

[6] AUTHORS, T. O. P. *SSL_read* [online]. Available at: https://docs.openssl.org/3.0/man3/SSL_read/.

[7] AUTHORS, T. O. P. *SSL_set_fd* [online]. Available at: https://docs.openssl.org/master/man3/SSL_set_fd/.

[8] AUTHORS, T. O. P. *SSL_write* [online]. Available at: https://docs.openssl.org/3.0/man3/SSL_write/.

[9] CRISPIN, M. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1* RFC 3501. RFC Editor, march 2003. Available at: https://doi.org/10.17487/RFC3501.

[10] EDDY, W. *Transmission Control Protocol (TCP)* RFC 9293. RFC Editor, august 2022. Available at: https://doi.org/10.17487/RFC9293.

[11] KERRISK, M. *Getopt(3) — Linux manual page* [online]. Available at: https://www.man7.org/linux/man-pages/man3/getopt.3.html.

[12] KERRISK, M. *Recv(2) — Linux manual page* [online]. Available at: https://www.man7.org/linux/man-pages/man2/recv.2.html.

[13] KERRISK, M. *Send(2) — Linux manual page* [online]. Available at: https://www.man7.org/linux/man-pages/man2/send.2.html.

[14] RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3* RFC 8446. RFC Editor, august 2018. Available at: https://doi.org/10.17487/RFC8446.

[15] RESNICK, P. *Internet Message Format* RFC 5322. RFC Editor, october 2008. Available at: https://doi.org/10.17487/RFC5322.