Secure Communication

October 8, 2023

1 Introduction

Client and server are defined in two files: `server.py`, and `client.py`, respectively. We developed them in `Python` programming language, using libraries `socket`, and `ssl`.

Server is a simple **synchronous** server which supports only one client connection at the time. It mimics the behaviour of professional servers to the minimal degree, meaning that it responds to the (`CLI`) commands issued by the client. Furthermore, since it is a `localhost` server, it can be used as a simulation of user's local machine - to some extent, since it only returns the result of the commands, meaning it cannot switch directories etc. If the requested command is unknown, it responds with the unknown exception message and continues to operate. This is achieved by running a `sub-process` that executes the command. Since all servers that are used in production environments include log files that can be monitored by system administrators - especially in the case of an error - we created that as well. It can be found in the file `".\log\server.log"`.

Both server and client use `socket library` to establish `TCP` connection between each other - socket. Furthermore, the established socket is wrapped around by the `TLS module` provided in the `ssl library`.

In order to use `SSL` secure channel we had to create our own `CA agency` - which we named `AIS_CA`. Then two private keys were created for `server`, and `client`. For each of them the `certificate - *.crt` was issued and signed by our CA. It was important that server certificate has its common name (CN) and subject Alt Name (SAN) set to equal its domain `'localhost'`. We were using command line `openssl` library to generate keys and certificates.

Since CA's certificate had to be trusted by our application we added it manually to both client and server code as trusted certificate using pre-defined function `load_verify_locations`. Then we loaded certificate and private key for both (server and client) correspondingly, using function `load_cert_chain`. Furthermore, since we wanted that both `client` and `server` present each other with corresponding certificate we added `ssl.CERT_REQUIRED` macro to both of them.

To verify that our channel is really secured, we captured the communication between server and client using wireshark. It can be found in file `".\log\communication.pcapng"`. From that we learned that our channel is indeed secured after handshake between them is finished - all data are encrypted and unreadable which shows that we have achieved `confidentiality`. Furthermore, **data integrity** is achieved automatically by TLS, since it appends `MAC` to the message and receiver calculates it again when it receives the message. If calculated MAC and appended MAC are not matching then the message has been corrupted and **integrity** has been violated. It is also not possible for a 3rd party to join secure channel without certificate issued by our own CA, thus, they cannot send messages.