Distributed Computing Project

Submitted By: Darragh Elbel T00211193

Computing with Software Development

Date Submitted: 07/12/23

Contents

[Introduction 3](#_Toc160883263)

[Short Message Protocol 3](#_Toc160883264)

[Abstract 3](#_Toc160883265)

[Introduction 3](#_Toc160883266)

[Functionality 3](#_Toc160883267)

[Log-on 3](#_Toc160883268)

[Short Message Upload 4](#_Toc160883269)

[All Message Download 4](#_Toc160883270)

[Specific Message Download 5](#_Toc160883271)

[Log-off 5](#_Toc160883272)

[Protocol-Wide Error Handling 5](#_Toc160883273)

[Secure Communications 5](#_Toc160883274)

[Application 5](#_Toc160883275)

[Application Layer 5](#_Toc160883276)

[Presentation Layer 6](#_Toc160883277)

[Conclusions 6](#_Toc160883278)

# Introduction

This project explores the processes of designing and implementing a secure message protocol. The protocol design should is implementation technology agnostic, outside of the requirement to implement secure communication via Secure Socket Layers (SSL). The application implementation will be done using Java and the stream socket API.

# Short Message Protocol

## Abstract

The Short Message Protocol (SMP) is a protocol to facilitate the secure saving and retrieval of short messages on a server, accessed via secure log-on based on a set of whitelisted user credentials.

## Introduction

The purpose of SMP is to provide a protocol which allows clients to securely connected, upload, and download messages from a server. SMP consists of 5 core functions, these being logon, the ability for a user to connect to the server using username and password combinations. Logoff, allowing users to disconnect from the server. Message upload, allowing users to upload and save a short message to the server. Specific message download, allowing users to retrieve a specific message. Finally, all message download, allowing users to retrieve all messages at once. The server should be capable of storing up to 999 short messages for every user and messages are identified by a message number from 1 to 999, 0 is NOT a valid message identifier. The server must allow for multiple clients to connect concurrently.

## Functionality

For each function requirement of the system (log-on, logoff, message upload, specific message download, bulk message download) the relevant section contains a description of the message format required, dictating allowed message formats. Each section also contains an error table of function-specific errors. Section insert section here contains an error table for protocol wide errors, these are errors that should be checked for before the server performs any parsing of messages.

### Log-on

System log-on should be facilitated by verifying username and password combinations, transferred to the server for authentication using SSL. Usernames and passwords should be a maximum of 256 bytes each. Only valid ASCII characters should be permitted. The following is an example log-on request in plain text. The maximum size of a LOGON request message is 519 bytes. 5 bytes are allocated to the LOGON prefix, 1 byte allocated to the following space, 256 bytes maximum for the username, 1 byte allocated to the following space, and finally 256 bytes maximum for the password.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5 Bytes** | **1 Byte** | **256 Bytes** | **1 Byte** | **256 Bytes** |
| LOGON | Blank space (ASCII 255) | Username | Blank space (ASCII 255) | Password |

On successful logon, the server should respond to the client with the message “USER AUTHENTICATED”. See error table 2.1.1 for details on authentication failure and other LOGON command errors.

#### 2.1.1 Error Table

|  |  |  |
| --- | --- | --- |
| **Code** | **Error Message** | **Description** |
|  | – INVALID AUTH FORMAT | Returned when a request is detected as being of type LOGON, but the message contains a greater number of bytes (>519) than is permitted. |
|  | – WRONG USERNAME/PASSWORD | Returned when user authentication fails due to an invalid username/password combination. |

### Short Message Upload

Messages upload via SMP are facilitated by the MSGUP command. Authenticated users should only be able to access their own messages, it should not be possible for any users to gain information on any other user of the system. A message upload packet should not exceed 1024 bytes, 6 of which are reserved for MSGUP and a following space.

|  |  |  |
| --- | --- | --- |
| **5 Bytes** | **1 Byte** | **1018 Bytes** |
| MSGUP | Blank space (ASCII 255) | Message Content |

On successful message, the server should respond to the client with the message “MESSAGE UPLOADED SUCCESSFULY”. See error table 2.1.1 for details on authentication failure and other MSGUP command errors.

#### 2.2.1 Error Table

|  |  |  |
| --- | --- | --- |
| **Code** | **Error Message** | **Description** |
|  | – INVALID MESSAGE FORMAT | Returned when a request is detected as being of type MSGUP, but the request is greater than 1024 bytes in size. |
|  | – NO MESSAGE | Returned when an MSGUP request does not contain message content. |

### All Message Download

Message download is facilitated by the MSGDL command. The MSGDL command is not accompanied by any other arguments and is 5 bytes in length. Downloaded messages should be returned in the format, if multiple message exist they should be concatenated together with a blank space “ “.

“MSG[MESSAGE NUMBER] – [MESSAGE CONTENT]”

For example, if the user as only uploaded one message, it should look like the following “MSG1 – This is my first message”. If the user has multiple message, it would look like this “MSG1 – This is my first message MSG-2 This is my first message”. When visible by the client, message numbers should NOT contain 0 prefixes. For example “MSG-01” is not a valid message identifier.

|  |
| --- |
| **5 Bytes** |
| MSGDL |

The maximum possible return for MSGDL if the user has stored 999 messages is 1,025,875 bytes, approximately 1 megabyte, so the client and server implementations should both support the communication of messages this size in a single transfer.

#### 2.3.1 Error Table

|  |  |  |
| --- | --- | --- |
| **Code** | **Error Message** | **Description** |
|  | – NO MESSAGES STORED | Returned when the logged-in user has no messages saved to the server. |

### Specific Message Download

Message download is facilitated by the SPMSG command. The SPMSG command is accompanied by a specific message id. For example “SPMSG 1” will return the first message uploaded by the user.

|  |  |  |
| --- | --- | --- |
| **5 Bytes** | **1 byte** | **3 bytes** |
| SPMSG | Blank space | 000-999 |

#### 2.4.1 Error Table

|  |  |  |
| --- | --- | --- |
| **Code** | **Error Message** | **Description** |
|  | – NO MESSAGES WITH PROVIDED ID | Returned when there is no message saved with the provided ID. |

### Log-off

System log-off is facilitated through the LGOFF command. The MSGDL command is not accompanied by any other arguments and is 5 bytes is length. Once logged off, the users session should be terminated.

|  |
| --- |
| **5 Bytes** |
| LGOFF |

#### 2.3.1 Error Table

SMP does not have any Log-off specific errors, failure to log-off due to not being signed in is facilitated by protocol-wide error handling.

### Protocol-Wide Error Handling

|  |  |  |
| --- | --- | --- |
| **Code** | **Error Message** | **Description** |
|  | – INVALID COMMAND LENGTH | Returned when the first 6 bytes of a message are not 5 characters of any type plus a blank space. |

### Secure Communications

Secure communications between the client and server should be facilitated through the use of SSL using a combination of public-key and symmetric encryption. The SSL implementation should apply to all messages send across the network to the server, including messages from a client that has not yet authenticated to ensure the secure transport of username and password combinations.

# Application Design and Implementation

## Design

### Objectives and Overview

The objective of this system design is to provide a comprehensive guide for the development of a server and client that can communicate via the Short Message Protocol. The system will be implemented in Java, and will allow for the concurrent connection of clients to a single server securely through TLS/SSL.

### Design Philosophy

As;dklfja;lskdjf

### Server-side Design

The following UML diagram demonstrates the relationship between the server side classes of the application. EchoServer3 is the entry point to the application, which is used to create EchoServerThreads for each connected client, this is a one to many relationship. EchoServerThread objects then in turn creates a new user session object when a client connects, which is a 1 to 0 to 1 relationship.

### A screenshot of a computer Description automatically generated

Figure 1 Server-side UML

### Client-side Design

As;dklfja;lskdjf

## Implementation

### Application Layer Implementation

As;dklfja;lskdjf

### Presentation Layer Implementation

As;dklfja;lskdjf

### Service Layer Implementation

As;dklfja;lskdjf

### User Manual

As;dklfja;lskdjf