Software Design Documentation
TauNet Client Application v1.0
Jacob Martin

---

1. Purpose

The purpose of this document is to describe and document the general design considerations and high-level architecture of the TauNet client application. This document is not intended to go into many low-level technical details.

---

2. Design Considerations and Usability

2.1. General Design Concept

In general, the design goals are to keep the application as simple, clean, accessible, and minimalistic as possible. With that in mind, the decision was made to make the application usable from the terminal using simple commands with command-line arguments. This way, the application can be utilized either from the hardware itself via keyboard/mouse/monitor, or also remotely through an SSH connection, improving accessibility and usability.

2.2. Separation of Responsibilities

There will be two primary commands for using the application. The first is a "send" command for sending messages to other TauNet client applications. This command will be called with arguments indicating the recipient and the message to send.

The second is a "receive" command for receiving messages from other TauNet client applications. This command will run continuously after it has been called, until an escape sequence or kill signal are issued.

Using separate commands allows for a user, if they so desire, to be able to send messages while not listening for responses or vice versa. Of course, they can also do both simultaneously if they desire as well, either using a split-screen terminal or multiple terminals.

3. High-Level Architecture

3.1. Encryption

As the CipherSaber-2 encryption algorithm will be leveraged both for the sending and receiving of messages, and it involves nested looping, the decision was made to implement the algorithm using C in order to make it as fast and efficient as possible. The algorithm is simple enough that it really shouldn't take longer to implement in C than it would take to implement in some higher level language. In fact, since the algorithm involves manipulating raw binary data, it may be even simpler to implement in C.

3.2. Client Table

The client table will be represented in plain text in a JSON file. An example client table JSON file will be included with the software with instructions for copying it to the correct location and filling it out with actual client details. Installing the client table is as simple as copying the example file to the correct (documented) location and filling it out while following the provided format.

The formatting of the client table JSON object is as follows:

- **key**: A string containing the encryption key being implemented by the TauNet network that we are utilizing.
- **username**: A string containing the username of the current user of the application utilizing the network (must be a member of the client table).
- **clients**: A dictionary where each key represents a unique username and each value a unique hostname or IP address of a client on the TauNet network.

The client table will be loaded upon each application command execution, and so it can be updated at any point in time.

3.3. Send and Receive Commands

Both of these commands will be implemented in Python. This decision was made as Python is a higher-level language with excellent network socket packages for establishing TCP connections and sending or receiving data.

3.3.1. Send Command

This command-line tool will accept two arguments: **username** and **message**. The username argument will indicate the username of the desired recipient of the message. The application will construct the proper headers, attach them to the message argument, and encrypt the whole resulting string.

The command will lookup the proper hostname or IP address from the client table and attempt to establish a TCP connection with that client. Upon successful connection, the encrypted bytes will be sent to the recipient and the command will finish gracefully. If the connection is unsuccessful, an error will be displayed to the user.

3.3.2. Receive Command

This command-line tool has no arguments. Upon running, the command will use the socket library to bind to the proper hostname and port and begin listening for incoming messages. The application will continue to listen until receiving a termination signal from the user.

When a message is received, it will be displayed in a user-friendly format like so:

```
12/05/15 12:39:48 <leng> Hello there.
```

If an improperly formed message is received, the improperly formed message in its entirety will be displayed to the user so they can determine further steps to take.