Software Design Document

TauNet

Josh Bucklin
CS300

Table of Contents

1.0 Purpose & Overview:

This document covers the basic architecture and design choices of the TauNet communication software. It covers why certain choices were made as well as provide pseudocode for the implementation of certain components.

This document is broken down into three main components:
1. System Overview – Covers what the system is supposed to accomplish
2. System Architecture – The overall view of how the system is pieced together
3. Design Details – This section covers each component of the software and breaks them down into smaller chunks providing explanations and pseudocode for each.

2.0 System Overview:

The TauNet communication system is aimed at providing its users with the ability to send and receive encrypted messages within a predefined network. This implies the basic functionality of this system is that it will allow it's users to send messages, receive messages and read them on the screen. Another implication of this functionality it is apparent that network and user information will need to be stored in some fashion.

3.0 System Architecture:

This section covers give an overview of how the system is pieced together. The system is laid out into two three main components:

3.0.1 Ciphersaber-2/RC4 – Both the server and the client will use this piece of the program to encrypt and decrypt messages. This will happen in one of two ways:

-The client will send a plain text message to the encrypt function which is then sent back to the client and sent to a receiving part.

-The server will send an encrypted message to the decrypt function which is then sent back to the server and displayed on the screen.

3.0.2 TauNet Server – This is how messages from the network will be received on a TauNet node. The server is in a constant state of listening for connections from other TauNet nodes. The state is altered by input from the user via a known command or if a message is received from another user. This implies that there is an interface that the user can interact with provided with.

The main dependencies of the server are:

- It will communicate with the Ciphersaber-2 component when decrypting messages

- It will get data necessary for operation from the local files with stored network data (User Table/Port Info)

- An internet connection in order to receive messages from other users in the network.

3.0.3 TauNet Client – This is how messages are sent to other servers in a TauNet network. The client is in a constant state of waiting for a user to send a message. The state is altered by the user inputting commands to send messages or view members of the network. This implies that there is an interface for the user to interact with.

The main dependencies of the client are:

- It will communicate with the Ciphersaber-2 component when encrypting messages.

- It will get data necessary for operation from the local files with stored network data (User Table/Port Info)

- An internet connection in order to receive messages from other users in the network.

It is important to note that the server and client will be separate pieces of software and will operate independently of each other on each TauNet node. However, the client depends on the server of another TauNet implementation to operate. It wouldn't be able to send messages otherwise.

The choice to separate the client and server was made because it eliminates unwanted side effects of the server and client using the terminal at the same time. It is possible to control these side effects with the use of locks and semaphores but that rapidly overcomplicates the design of the system and so was avoided.

4.0 Design Details:

This section outlines how each component of the TauNet software will operate. It attempts to break down each component into pseudocode in order to fully understand its operation.

4.1 TauNet Server:

The main functions of the TauNet server include:

4.1.1 **Function**: Start the Server
Description: Describes how the server is started. It's necessary to create a socket and enter an infinite loop waiting for connections from other users.
Pseudocode:

```
Create a new thread for the server to run on
Use function 4.1.2 to load program data
Get Encryption key from user or a file
Create a socket to receive TCP connections and bind it to the port # loaded from function 4.1.2
        If the binding fails exit the program immediately
While the program is running:
        Wait for incoming connections
        If a connection is received use function 4.1.3
```

4.1.2 Function: Load Program Data

Description: This function will load needed data from a local file(s) including user table, port info, version #, user name, and message size.

Pseudocode:

Open file(s) containing program data.
While there are still usernames and IP/Domains to read:
-Read each username and IP/Domain into a local data structure that allows easy lookup. If using python a dictionary is recommended. If any name doesn't meet protocol standards discard it.
-As each IP is read in in also store it in another data structure that will later be used to lookup IP's to verify valid incoming messages. If using python a list is recommended.
While there is still program data to read:
-Read each additional piece of data (port info, version, etc..) storing each in variables.
Once all data has been read then close the file(s).

4.1.3 Function: Receive a Message

Description: This function will handle incoming data from other TauNet users.

Pseudocode:

If a message is received store it in a variable called ciphertext
If the ciphertext is empty
Discard the message
Return to listening
Create a new thread that calls function 4.1.4
Resume listening for more messages

4.1.4 Function: Display a Message

Description: This function will handle decrypting and displaying messages from other nodes.

Pseudocode:

Use Ciphersaber-2 to decrypt the message
Acquire a locking device that prevents other functions from printing to the terminal
If the message came from an unknown user or a different version of TauNet
Display a warning to the user
Display the message with a trailing blank line
Release the locking device
Close the thread.

4.1.5 Function: Display a Users of the Network

Description: When the user types an appropriate command all members of the network should be displayed on the screen. This will utilizes thread locking mechanisms to prevent multiple functions from printing to the screen at the same time.

4.1.6 Function: Change Encryption Key

Description: This is a function that allows the user to change the key during run time. This is a function that I personally decided to add.

4.2 TauNet Client:

The TauNet client will consist of a menu that the user can use to choose to send messages, view

members of the network, change the key and exit the program.

The main functions of the TauNet client include:

4.2.1 **Function**: Send Messages to TauNet Nodes
Description: This function enables the user to be able to send messages to other users within their network.
Pseudocode:
Use function 4.2.2 to get the receive for the message
Use function 4.2.3 to get the message to send
Encrypt the message
Create the header for the message as specified in the TauNet protocol v0.2
Add the headers to the message
Create a connection with the receiving user
If a connection is not made within 5 seconds
    Notify the user that the receiver is unavailable
    Return to the main menu
Transfer message to the receiving user

4.2.2 **Function**: Get a Receiver for the Message
Description: This function uses function 4.2.4 to display the users within a the network and ask the user to enter the name of the user they would like to send a message to.
Pseudocode:
Display users of the network using function 4.2.4
Ask user to enter a username of who they would like to message
If the username is not valid
    Repeat previous step
If the username is equal to cancel
    Return to main menu of program
Return the chosen username

4.2.3 **Function**: Get a Message from the User
Description: This function gets the message to send to another user. Making sure that it conforms to the protocol specifications.
Pseudocode:
Display a prompt to the user to enter a message
If the message is too long
    Tell the user it's too long
    Return to previous step
Return the message

4.2.4 **Function**: Display the Users of the Network
Description: When the user types an appropriate command all members of the network should be displayed on the screen. This will utilizes thread locking mechanisms to prevent multiple functions from printing to the screen at the same time.

4.2.5 Function: Change Encryption Key
Description: This is a function that allows the user to change the key during run time. This is a

function that I personally decided to add.

4.2.6 **Function**: Load Program Data

Description: This function will load needed data from a local file(s) including user table, port info, version #, user name, and message size.

Pseudocode:

Open file(s) containing program data.
While there are still usernames and IP/Domains to read:
    -Read each username and IP/Domain into a local data structure that allows easy lookup. If using python a dictionary is recommended. If any name doesn't meet protocol standards discard it.
While there is still program data to read:
    -Read each additional piece of data (port info, version, etc..) storing each in variables.
Once all data has been read then close the file(s).

4.3 Ciphersaber-2/RC4:

This section of my program is based off pseudocode that was created by Bart Massey and so I don't really see much point to elaborate on pseudocode for this section. Instead I just give a brief description of what is expected of each function.

The original documentation/pseudocode can be found here:
https://github.com/BartMassey/ciphersaber2/tree/master/pseudocode

4.3.1 **Function**: RC4

Description: This function generates a key stream that the encryption and decryption functions use to encrypt/decrypt their messages. The length of the key stream corresponds to the length of the message. This function expects a message length, # rounds of key scheduling, and a key. Further implementation details can be found in the linked repository.

4.3.2 **Function**: Encrypt

Description: This function encrypts messages using the key stream generated by RC4. It behaves roughly in the following way: It takes a message, # rounds, and a key. It then adds a random IV to the end of the key before passing the necessary info the RC4. The cipher text is generated by getting the key stream from the RC4 function and xor-ing it with the original message + iv. The cipher text is returned. Further implementation details can be found in the linked repository.

4.3.3 **Function**: Decrypt

Description: This function decrypts messages using the key stream generated by RC4. It behaves roughly in the following way: It takes a message, # rounds, and a key. It then removes the iv from the cipher text and adds it the key. The necessary info is passed to the RC4 function which returns a key stream. The plain text is generated by xor-ing the key stream with the cipher text. The plain text is returned. Further implementation details can be found in the linked repository.