

TauNet

Software Design Document

Copyright © Gregory Gaston 2015

Table of Contents

1. Introduction

- a) Purpose**
- b) Scope**
- c) Overview**
- d) Reference Material**
- e) Definitions and Acronyms**

2. System Overview

3. System Architecture

- a) Architectural Design**
- b) Decomposition Description**
- c) Design Rationale**

4. Data Design

- a) Data Description**
- b) Data Dictionary**

5. Component Design

6. Human Interface Design

- a) Overview of User Interface**
- b) Screen Images**
- c) Screen Objective and Actions**

7. Requirements Matrix

8. Appendices

1. Introduction

a) Purpose

This Software Design Document describes the system design of TauNet, a system that delivers secure messages to its users via the internet.

b) Scope

The goal of this software is to provide users a secure way to transmit messages across the internet that will be difficult for any unwanted sources to read. The software will also be designed to function on a trusted piece of hardware based on a secure platform.

c) Overview

Over view of this document's organization

d) Reference Material

Optional: Links to source material for document.

e) Definitions and Acronyms

TauNet – The system

TauNetwork – A network of users connected with TauNet

TauNode – A single system on a TauNetwork

2. System Overview

TauNet will deliver a secure platform for users to anonymously and privately send simple text messages across the internet. The user will be able to send messages to TauNodes on a TauNetwork that they are connected to. The TauNetwork is set up by users sharing their individual IP Addresses, User Addresses, and a shared passkey privately using a different means of communication, face to face is the recommended method. Once the TauNetwork is set up when a user sends a message to another

user the system will encrypt the message before sending it, and upon arrival the message will be decrypted by the other user.

This system is needed due to the increasing amount of surveillance being conducted by the nations of the world and the involvement of major corporations in support of their surveillance. This problem requires the introduction of a secure messaging platform that sends encrypted messages over a network of trusted devices that do not rely on software provided by governments or large corporations to function.

3. System Architecture

a) Architectural Design

The system will be broken down into 5 major subsystems.

1. Main

The Main subsystem will contain the main program loop, as well as the basic methods for sending and receiving messages. The Main subsystem makes calls to the other subsystems in order to achieve overall system functionality.

2. Encryption

The Encryption subsystem will contain the methods for encrypting a message. This system interacts with Main by taking in messages and returning them either encrypted or decrypted.

3. Protocol

The Protocol subsystem contains methods for composing, and reading messages based on a standardized arrangement. This system interacts with main by composing sent messages before encryption and decomposing received messages after decryption.

4. Message History

The Message History subsystem contains the methods that store recent messages in memory. This subsystem interacts with main by taking a message and storing it in to a list of messages recently received.

5. TauNet

The TauNet subsystem handles the methods that set up the TauNetwork and store it for later use. The TauNet subsystem interacts with main to house a list of user names and IP addresses for TauNodes, as well as the methods for storing that list when the system goes off line.

b) Design Rationale

The system was decomposed in to these five subsystems in order to make the project more manageable. All subsystems, except for Main, are easily tested independently and can be incrementally added to Main without requiring the other subsystems to operate. This method is especially useful when making modifications to the subsystems. Whenever a subsystem is modified the only other system that may need modification is main, to incorporate the new functionality.

4. Data Design

a) Data Description

The list of TauNodes is stored as a linear linked list of users. Initially the user will type the required information in to prompts, and the system will store that information in to a linked list of nodes, thereby setting up the TauNetwork. The system will also store that information in to a text file that will be stored on program exit. In future executions of the program users will be able to load the list of users from that file to set up the TauNetwork.

The list of messages is stored as a doubly linked list. New messages will be pushed

to the top of the list, with the oldest message being the last on the list. Only a certain number of messages will be allowed to be stored due to memory concerns. When the maximum amount of messages is surpassed oldest message is discarded.

5. Component Design

Main

```
Send_Message(user_name, message):
```

```
    m = compose(Message) #Compose
```

```
    enc_m = encrypt(m, rounds, key) #Encrypt
```

```
    IP = Search_user(user_name) #Retrieve IP
```

```
    send(IP, enc_m) #Send Message
```

```
Recieve_Message():
```

```
    while 1:
```

```
        enc_m = recieve() #Recieve Message
```

```
        m = decrypt(enc_m, rounds, key) #Decrypt Message
```

```
        message = decompose(m) #Decompose Message
```

```
        print(message) #Display received message
```

TauNet

```
add_user(user_name, ip):
```

```
    if not valid_user(user_name): return False #make sure it is properly formatted
```

```
    new_node = user_node(user_name, ip) #create a new user node
```

```
    head = push(new_node) #Push a new node on to the stack of nodes
```

```
    return True #Report Success
```

```
search_users(self, user_name):
```

```
if self.user_name == user_name:

    return (self.user_name, self.ip)

if self.next: return self.next.search_users(user_name)

return None
```

Write_File(filename): #Read file is similar but reads instead of writes

```
f = open(filename, 'w+')

f.write(key + '\n' + my_user_name + '\n')

current = head

while current

    if current.user_name and current.ip:

        f.write(current.user_name + '\n' + current.ip + '\n')

    current = current.next
```

Message History

```
add_message(message)

if head:

    new_message = mess_node(message)

    head = push(new_message)

else:

    head = mess_node(message)

m_count = m_count + 1

if m_count > MAX_MESSAGES:

    pop_tail()

    m_count = m_count - 1

return True
```

6. Human Interface Design

a) Overview of User Interface

The system will be presented to the user as a simple command prompt. The user will input commands that represent different functions similar to how they would issue commands to command line.

Examples:

TauNet >> – Example command input line

@user message – Sends user a message

+user ip – Add a user and their ip address to the list of users.

When the user receives a message the message will be displayed and the command prompt will again be displayed as if there was no disruption to the user's operation.

Example:

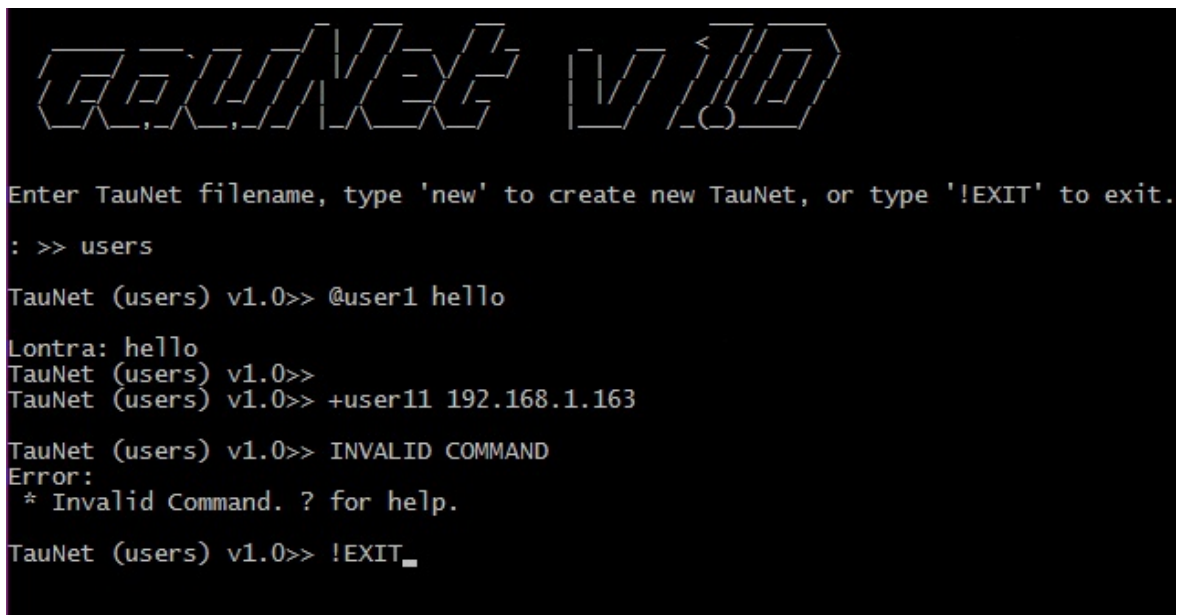
TauNet >>

Sender: Message from Sender

TauNet >>

All required functionality for sending messages will be implemented with commands to the TauNet command line, when the user inputs a command that is unexpected the system will alert the user and display an error message with an option to request help. All requirements for receiving messages will be implemented with how the message is displayed to the user once a message is received.

b) Screen Images



```
TauNet v1.0
Enter TauNet filename, type 'new' to create new TauNet, or type '!EXIT' to exit.
: >> users
TauNet (users) v1.0>> @user1 hello
Lontra: hello
TauNet (users) v1.0>>
TauNet (users) v1.0>> +user11 192.168.1.163
TauNet (users) v1.0>> INVALID COMMAND
Error:
* Invalid Command. ? for help.
TauNet (users) v1.0>> !EXIT_
```

7. Requirements Matrix

SRS Requirement	How requirement is satisfied
III.2.a	send_message(), receive_message()
III.2.b	encrypt()
III.2.c	Socket.settimeout(8)
III.2.d	at() will have exception handling for unsent messages
III.2.e	user_list class that stores users as a LLL
III.2.f	TauNet Protocol Document specifies messages include the name of the user that sent the messages. receive_message() will also display the user name of the sender.
III.2.g	Software 'read_me' will recommend the user use Raspberry Pi device with the Rasbian Operating system installed.
III.3.a	main() will be written to be used in terminal.
III.3.b	Each TauNode will house the information needed to connect to other nodes. The all commands the user can issue will be tested for stability.
III.3.c	A simple efficient encryption method will lbe used and Socket.settumeout(8) will only allow 8 seconds before a message is determined to be unsendable.
III.3.d	A command for a help menu will be included in the program.
III.3.e	This requirement reinforces III.2.g and III.3.a

III.3.f	Message_List Class saves the recent messages and # command redisplay messages.
----------------	---