Assignment 1 Report

Vincent Mai
Justin Nguyen
Malik Eleman

Introduction

The primary task of this assignment was to create a client and a server over a TCP connection, such that the client will extract, in real time, a tweet containing a certain question, that would get encrypted into a payload when transferred to the server. The server will then decrypt and verify the question so that the question can be spoken, solved, and then sent back to the client for the answer to be spoken.

Design and Implementation

In order for this task to be completed, the client and server were first initialized to see if the TCP connection is valid. Secondly, both the client and server must run with a given set of parameters passed in through the command line. Thus, the next step was to parse and validate the correct type and number of arguments within the list to extract the server port, socket size, and server IP.

After initializing the client and server along with their parameterized calls, the tweet extraction was implemented with the ‘Tweepy’ Module. Since the requirements were to implement it as a stream API, it was difficult for us to figure out that the tweet extraction extracted one tweet in real-time before moving forward. This was probably the most difficult task for us.

The tweet was then parsed so that only the question was extracted. The question was then encrypted and hashed into a payload tuple to be sent over to the server. The server will receive this tuple and hash the encrypted message to verify that the hashes match and that nothing was ‘modified’. Then, the IBM Watson API was used for the ‘text to speech’ operation and the Wolfram Alpha API was used to provide an accurate answer to the question. The answer was then returned to the client with the same encryption and verification method to be sent to IBM Watson to speak the answer.

Overall, we were able to easily grasp the concept of this assignment, however, we did run into a blocking issue that was caused by a misinterpretation of Tweepy’s Stream API such that we didn’t know the tweets were extracted real-time and not through ‘history’.

Additionally, we, as a team, faced a roadblock where we were unable to set up the client and server to receive multiple queued tweets, such that the problem resided with the client closing its connection and the server trying to reconnect to the client when the session was established already. We resolved this by removing the close statement which allows the client to run until forced to stop and only enabled the server to look for the client once.

Libraries and Modules Imported

import ServerKeys as sk

- ServerKeys is a python file that has all the necessary keys that would be used on the server side. It has the keys and authentication access for both IBM watson api and wolfram alpha api.

import ClientKeys as ck

- ClientKeys is a python file that has all the necessary keys that would be used on the client side. It has the keys and authentication access for both Twitter API keys and IBM Watson's API keys.

import sys

- sys, or short for the system library of Python, is used to deal with the python interpreter and command line. This was used to obtain the arguments (server IP, server port, and socket size) over the command line and parse it into correlating variables.

import socket

- Provides access to the BSD socket interface, which is used in the assignment to connect the client with the server through TCP.

import tweepy

- Tweepy is a library that is used to interact with the Twitter API. In this assignment it was primarily used to extract the tweet using StreamListener from the Stream API

import re

- Short for the regex, or regular expressions, library in Python. Used primarily to parse the extracted tweet into a string that contains only the question.

import pickle

- Pickle is a library that allows python object hierarchies to be converted into a byte stream as well as the inverse where a byte stream can be converted to the original python object. This was used in the project for sending a payload holding a key, cipher text, and a checksum as a single package.

import hashlib

- Hashlib is a module that allows you to create secure hashes and message digest algorithms. In this project, we specifically used the md5hash in order to secure our data. With the use of creating keys and using keys to decrypt cipher text, as well as creating a checksum to ensure data was properly sent over.

import simpleaudio as sa

- Simple Audio is a module that allows you to play anytype of audio files (ex. wav, mp4, m4a, etc.) In this case, the audio file created from IBM watson can be played on the computer and listen to dynamically as soon as the audio file is created.

import wolframalpha

- Wolfram alpha is a module that allows you to interface with the wolframalpha api. It allows you to easily create a client object that can be queried with the question you want and generates an answer.

from os.path import join, dirname

- Os join and dirname are functions that allow you to combine and find names of files including their directory. This was used during the process to create the audio files from IBM watson. This ensures that audio files are created locally at the directory the server or client file are called.

from cryptography.fernet import Fernet

- Cryptography.fernet is a module that allows us to create a symmetric/decryption and its corresponding key. This was used to make sure the question and answer are properly encrypted before sending over.

from ibm_watson import TextToSpeechV1, ApiException

- IBM watson modules allow us to use the IBM watson api in order to convert a string text to an audio file holding audio spoken text. This was used with texttospeechV1 while api exception was a error exception to be checked if the string could not be converted to an audio file.

from ibm_cloud_sdk_core.authenticators import IAMAuthenticator

- IBM cloud sdk module is a conjunction api with IBM watson text to speech that allows developers to authenticate their access to the IBM watson modules and apis.

Team Contributions

Vincent Mai:

- Created Github and README Task List
- Initialized Client
- Validate and Parsed Command Line Arguments
- Primarily work with Tweet Extraction using Tweepy's Stream API
- Helped formulate encryption, decryption, and MD5 Hash algorithm
- Wrote 'Introduction' and 'Design and Implementation of Report' and some of the used modules' description

Justin Nguyen:

- Created dev accounts for IBM watson, Wolfram alpha
- Worked/supported using tweppy module in order for real time tweet to be accepted
- Worked on creating initial server and client files and logic
- Used Watson API to create audio file in local directory of question, also created the same logic to be used for answer
- Created logic that plays the audio file in respected server and client files
- Created logic for payloads, creating key, decrypted cypher text, and check sum
- Created checking for check sum to ensure payloads are send correctly
- Encrypted and decrypted payloads
- Wrote most parts of what modules that were included and why they were included.

Malik Elemam:

- Created twitter dev accounts and python apikey files
- Created logic for wolframalpha to get answer from question string
- Integrated wolfram alpha portion into server code
- Tied together client and server files to complete the answer creation and sending
- Added print statements for grading
- Fixed style issues