California State University of Long Beach

Blank
An Encrypted Chat Application

By Cyberguard

Sotheanith Sok and Hayat Ahmed

CECS 478

Dr. Mehrdad Aliasgari

December 13th, 2018

Table of Content

I. Requirement

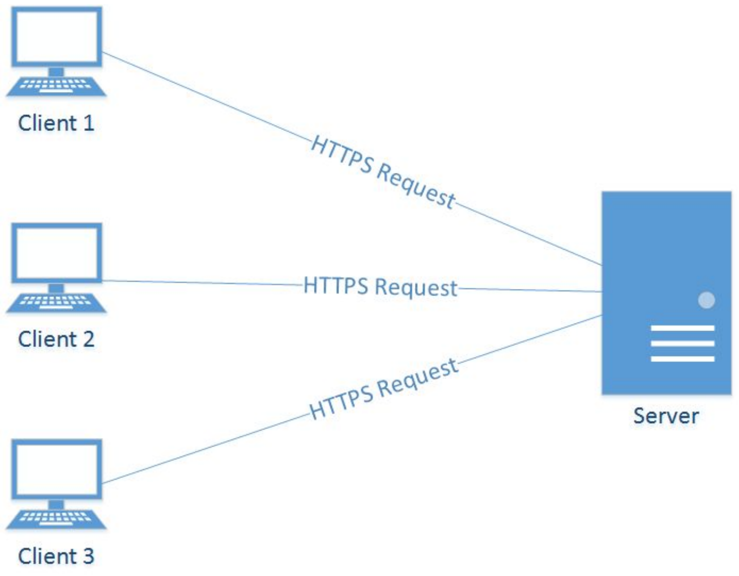
In today’s internet-connected world, information has become the most precious commodity. It can help to propel companies especially tech companies that offer free services, into a multi-billion dollars business empire and it is also the catalyst that leads to the downfall of many career politicians and powerful people. Unfortunately, the drive to mine and obtain users information by powerful entities has led to an erosion of users personal privacy especially when it comes to direct communication between individuals such as messages and emails. This application is designed to alleviate and recover some of the lost privacy of users by allowing users to send messages with confident that only they and their intended recipients can read the message. In order to complete such a goal, this application needs to satisfy four main criteria:

1. The client must encrypt each message in a way that only the intended recipient of such a message can decrypt them.
2. The communication link between a client and the server must be secure.
3. The server should not contain any information that enables it to decrypt messages that it supposes to distribute to clients.
4. The client must not be a web-based application as those type of application relies on the server for resources and execution behaviors.

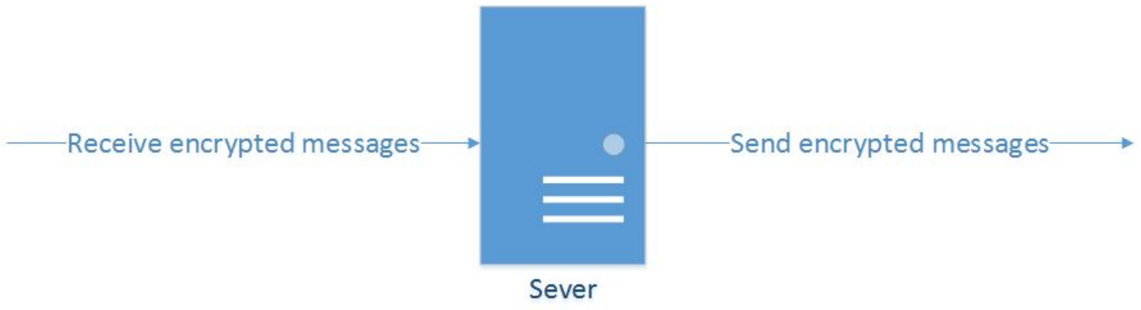
In its current iteration, this application satisfies the four criteria with some caveats. Furthermore, this application is not an end-to-end encryption chat due to the fact that it lacks features that are essential to an end-to-end encryption chat. A detail looks into the application design, flaws, and possible improvements will be included in sections below.

II. System Overview

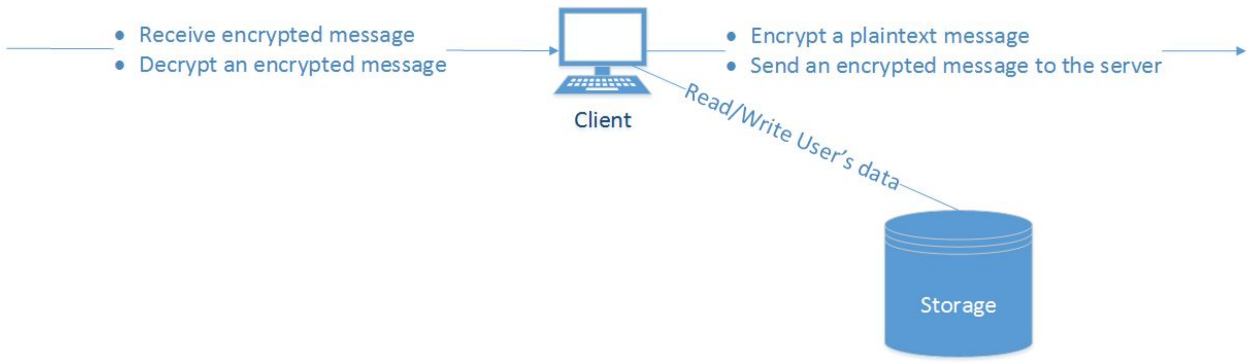
This application can be divided into three main components with each component is functionality independent of and serves different purposes than other components.



The first component is the server. It is the central entity of our system and without it, this system would not work. The main purpose of the server is to receive and distribute encrypted messages to its intended recipient. In order to establish some form of authentication and accessing privilege, the server requires that each user registers himself or herself with it and then, it will provide temporary access token to users when they authenticate themselves to the server.

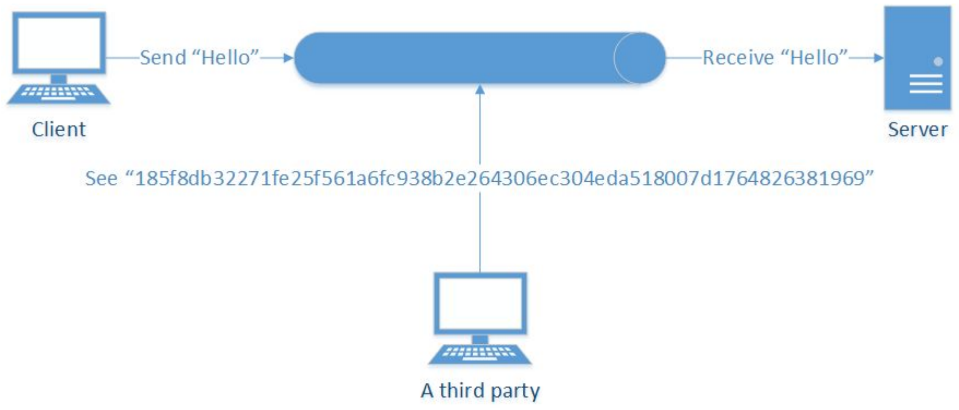


The second component is the client and it's an entity with multiple responsibilities. To start with, the client is responsible for the encryption and the decryption of messages send and receive from the server. It is also tasked with the management of users' sensitive and insensitive information. However, it does not authenticate the user as that responsibility should be held by the server.



• Receive encrypted message
• Decrypt an encrypted message
Client
• Encrypt a plaintext message
• Send an encrypted message to the server
Read/Write User's data
Storage

The third and final component of this system is the communication channel. Its main responsibility is to secure the transportation of data between the server and a client to prevent a third party from obtaining any useful information by monitoring the traffic.



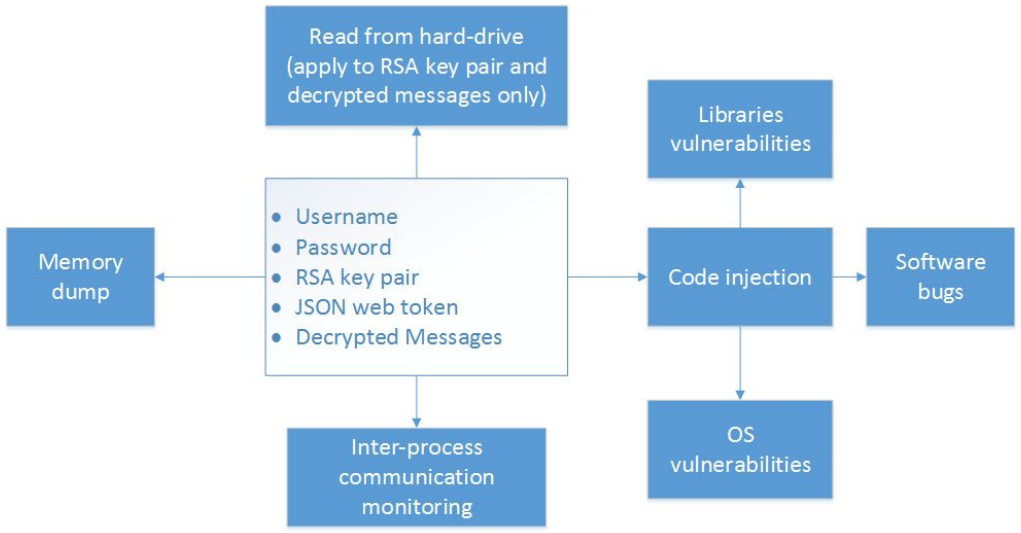
Send "Hello"
Receive "Hello"
Client
Server
See "185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969"
A third party

III. System Design and Analysis

A secure system must be designed from the ground-up to be secure and any security that was added after the completion of a system is doomed to fail. In order to achieve security, it is important that each design decision of a system goes through a complete analysis. Since the three components of this system can function independently of the others, the design and analysis of each component should also be independent.

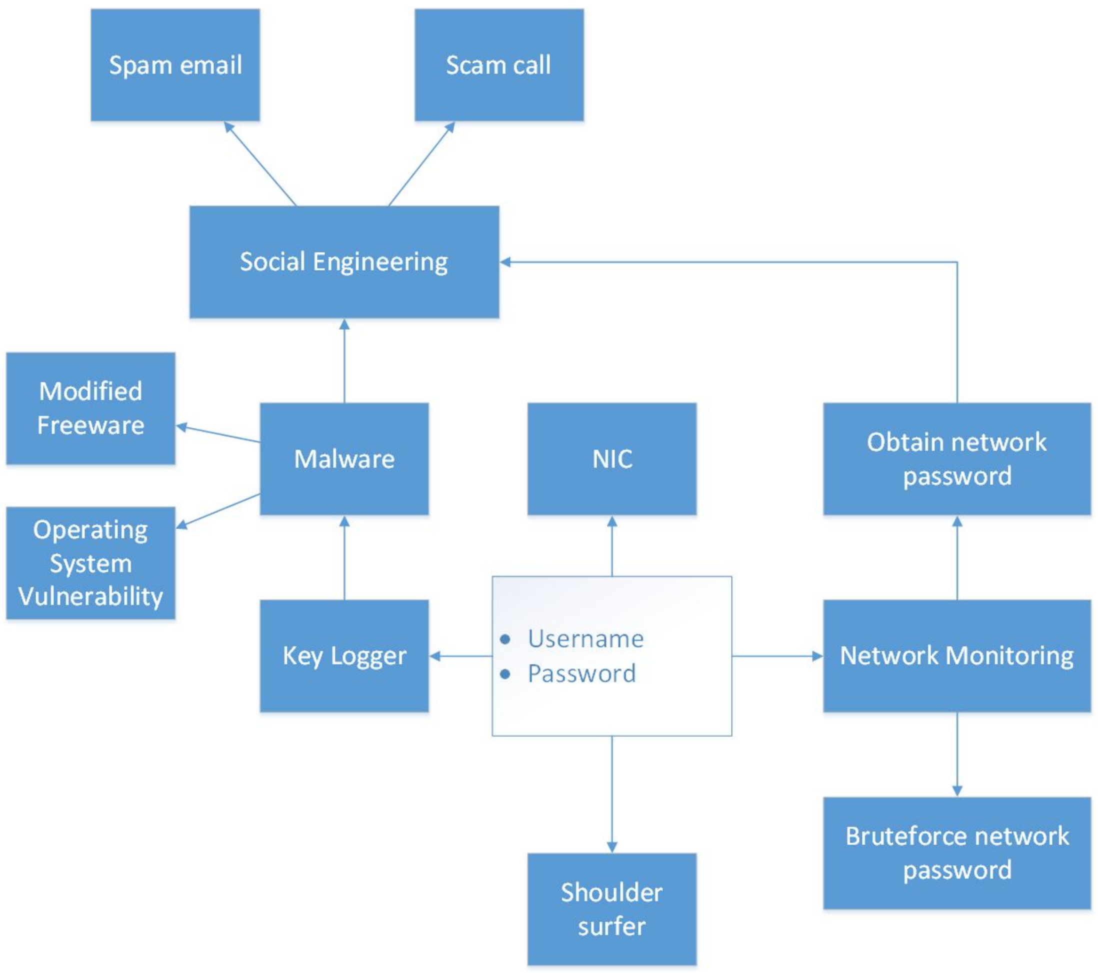
1. Client

- Assets:
  - Username
  - Password
  - User's RSA public key
  - User's RSA private key
  - User's JSON web token
  - User's known-public-key of other people
  - Decrypted messages
- Stakeholders:
  - User
- Adversaries:
  - An active outside adversary
  - An active inside adversary
- Attack Surfaces:
  - Combo 1:
    - Assets: Username, password, RSA key pairs, JSON web token, decrypted messages
    - Stakeholders: Users
    - Adversaries: An active inside adversary
    - Diagram:


Read from hard-drive (apply to RSA key pair and decrypted messages only)
Libraries vulnerabilities
Username
Password
RSA key pair
JSON web token
Decrypted Messages
Memory dump
Code injection
Software bugs
Inter-process communication monitoring
OS vulnerabilities

    - Research:

        - Code injection attacks rely on vulnerabilities in the operating system, libraries, or software itself in order to introduce malicious code into and modify the behavior of the software. The only viable preventions of such an attack are to validate every input into the system and ensure that the operating system and libraries are up to date.
        - Users' data in the client can be grouped into two broad categories: persistent data which stores on hard drives and temporary data which stores in the volatile memory. The general consensus on the security of temporary data is that the protection of such data is handled by the operating system itself. Thus, if the operating system gets compromised, then an adversary can access temporary data by taking a snapshot of the volatile memory. However, it is possible to improve the security of persistent data through some form of encryption.
    - Solution:
        - In order to prevent code injection attack, the client will validate all inputs into the system and turn off parsing of such inputs.
        - In order to improve the security of stored data, the client will encrypt all data that it needs to store on hard drives with a user's password. The encryption process will be a combination of the PBKDF2 and the AES-256-GCM.
    - Cost Analysis:
        - The decision to implement the password based encryption has a minimum effect on the development time and cost of the project as the PBKDF2 and the AES-256-GCM are natively supported by Node.js.
        - However, there is a foreseeable effect on the initial startup time of the client. As a user successfully logging in, the client will try to decrypt such user's data and load it in. This is a one-time operation that occurs every time a user login into the client. However, if the size of a user's data is large, then it will take a noticeable amount of time to decrypt the data.
- Combo 2:
    - Assets: Username, password
    - Stakeholders: User
    - Adversaries: An active outside adversary
    - Diagram:



- Research:
  - Keyloggers attack is a type of attacks on users' sensitive information especially username and password by capturing those pieces of information during the input process. The only way to counter such an attack is to rely on a trusted third party software for protection or only enter sensitive information in a clean operating system. Unfortunately, the solutions described above is not worth the cost of implementation in comparison to the security they provided.
  - Another way of obtaining users' sensitive information is by monitoring the network of such users. Even if the connection between two entities is protected with an encryption such as TLS 1.3, there is a small gap between when HTTPS request occurs and when data get encrypted where data can be read in plaintext. Unfortunately, there is not a standard protocol to protect against this type of attack and thus, it is up to an operating system to enforce access control and prevent the read of network data by an unauthorized individual.

2. Server
   - Assets:
     - Server Signature Key
     - Users' sensitive data including salt and hashed password

- ○ User's sensitive raw data including plaintext password
- ○ Encrypted Messages
- ○ SSH key
- ○ Server's availability

➢ Stakeholders:

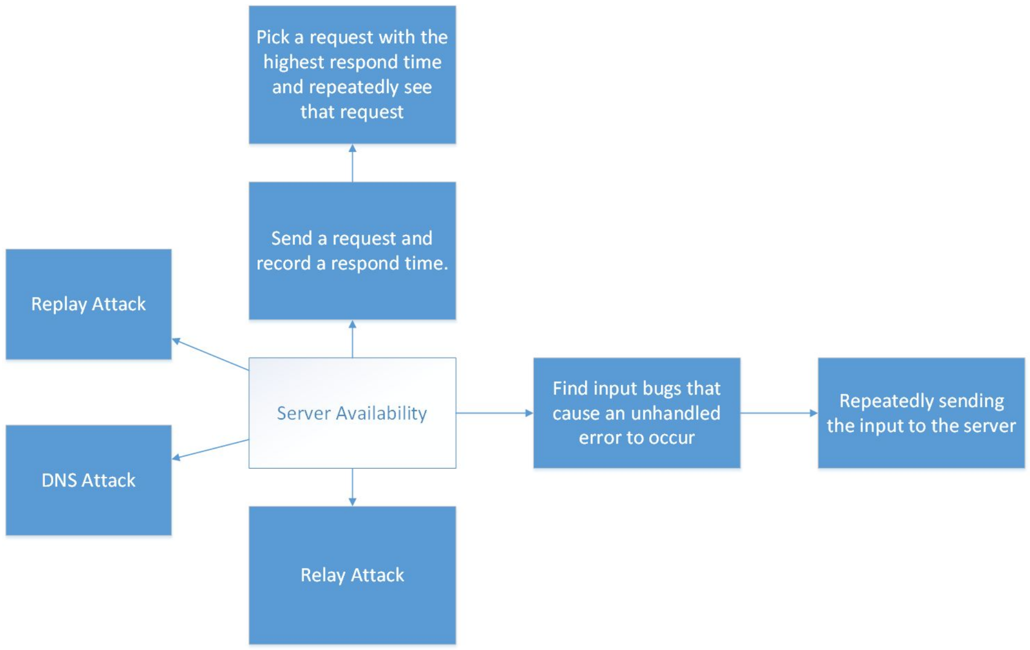
- ○ Users
- ○ Developers

➢ Adversaries:

- ○ An active outside adversary
- ○ An active inside adversary

➢ Attack Surfaces:

- ○ Combo 1:
  - ■ Assets: Server's availability
  - ■ Stakeholders: Developers
  - ■ Adversaries: An active outside adversary
  - ■ Diagram:

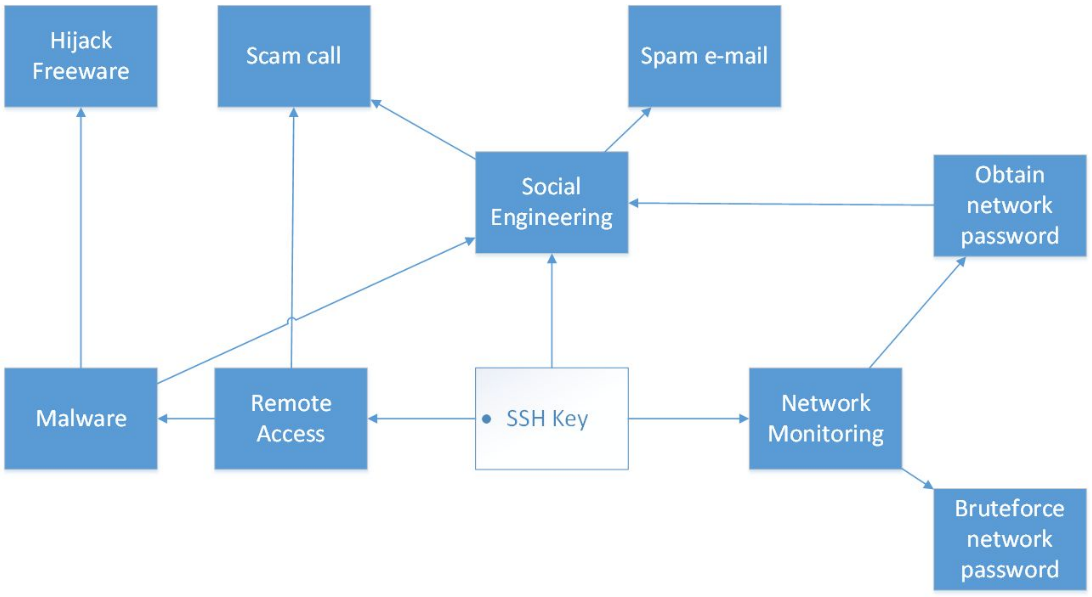

Pick a request with the highest respond time and repeatedly see that request
Send a request and record a respond time.
Replay Attack
Server Availability
Find input bugs that cause an unhandled error to occur
Repeatedly sending the input to the server
DNS Attack
Relay Attack

  - ■ Research:
    - ● The only ways to limit or prevent a replay or replay attack is by validating each request using some unique factors such as sessions' ID, one-time password, nonces and MAC, or timestamps.
    - ● Input bugs that lead to a system failure can be prevented by validating all incoming data and keeping the operating system and libraries up to date.
    - ● The DDOS attack is an attack on the system availability by overwhelming a system with a lot of workloads. There is no protocol that prevents such an attack as it is the nature of the server to be available to everyone. However, it is possible to lessen the

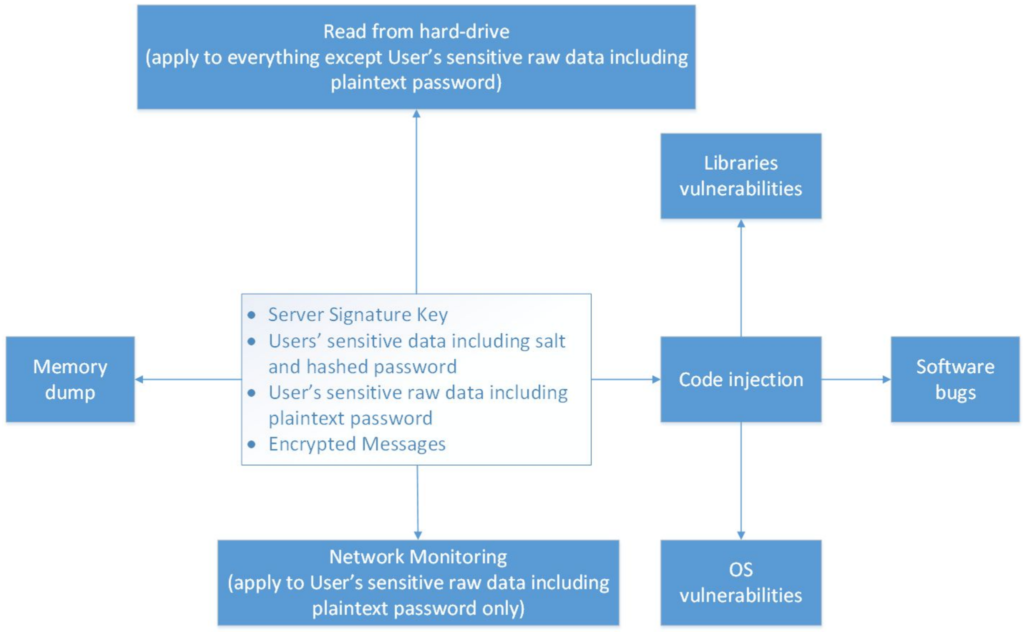
severity of the attack by introducing an access control where only authorized can make a "costly" request to the server.

- Solution:
  - The server will utilize JSON web token to ensure that it will only accept a "costly" request from authorized users.
- Cost Analysis:
  - Understandably, the introduction of JSON web tokens creates an overhead cost where the server has to verify the token attached to the request before it handles such request. In addition, it also introduces a new asset in the form of a signature key that needs to be protected as this key is used by the server to sign new tokens and verify existing tokens. However, the overhead cost of JSON web token creation and validation are small in the comparison to the cost of handling bad requests.

- Combo 2:
  - Assets: SSH key
  - Stakeholders: Developers
  - Adversaries: An active outside adversary
  - Diagram:



  - Research:
    - The protection of the SSH key is the responsibility of developers and it is up to them to ensure that they are not compromised.
- Combo 3:
  - Assets: Server Signature Key, users' sensitive data including salt and hashed password, user's sensitive raw data including plaintext password, encrypted Messages
  - Stakeholders: Developers
  - Adversaries: An active inside adversary
  - Diagram:


Read from hard-drive
(apply to everything except User's sensitive raw data including plaintext password)
Libraries vulnerabilities
Server Signature Key
Users' sensitive data including salt and hashed password
User's sensitive raw data including plaintext password
Encrypted Messages
Memory dump
Code injection
Software bugs
Network Monitoring
(apply to User's sensitive raw data including plaintext password only)
OS vulnerabilities

- Research:
  - Code injection attacks rely on vulnerabilities in the operating system, libraries, or software itself in order to introduce malicious code into and modify the behavior of the software. The only viable preventions of such an attack are to validate every input into the system and ensure that the operating system and libraries are up to date.
  - Protection against memory dump reading, network monitoring, and file access on a physical drive is the responsibility of an operating system. The operating system needs to enforce access control so that each process only has a minimum access to resources for its functionality.
- Solution:
  - In order to prevent code injection attacks, the server will validate all input that it receives and turn off parsing of user's input.
  - Additionally, all processes will only have a minimum level of access privilege so that no process can monitor or interfere with another process.
- Cost Analysis:
  - There is no extra cost in implementing access control as such system is natively supported by an operating system.

3. Communication Channel

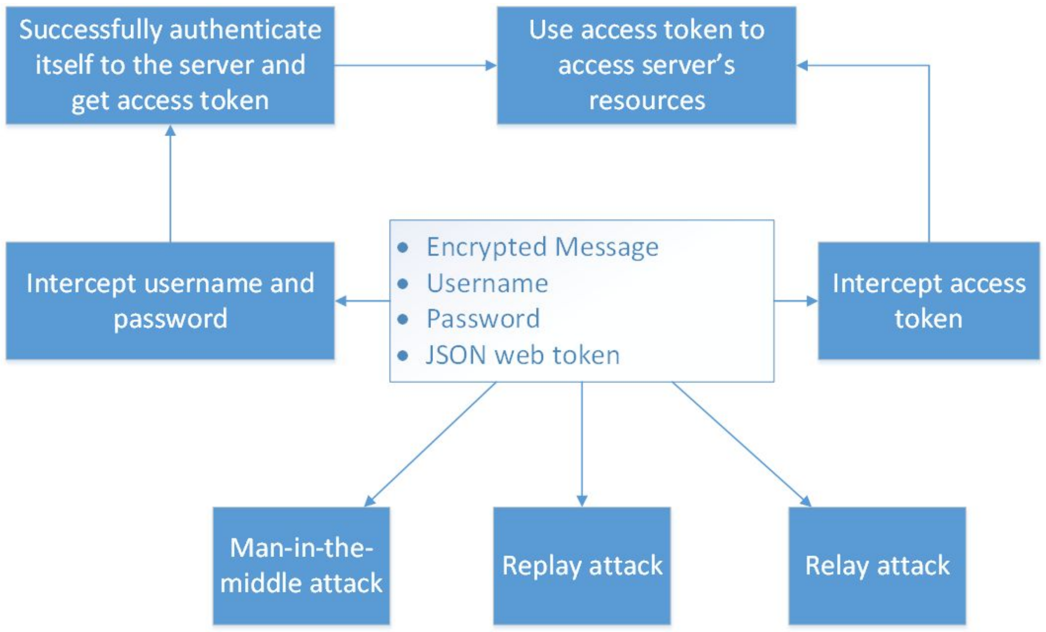
➢ Assets:
  - Encrypted Message
  - Username
  - Password
  - JSON web token

➢ Stakeholders:
  - Users

➢ Adversaries:

- ○ A passive eavesdropping adversary

➢ Attack Surfaces:
- ○ Combo 1:
  - ■ Assets: Username, password, JSON web token, Encrypted Message
  - ■ Stakeholders: Users
  - ■ Adversaries: A passive eavesdropping adversary
  - ■ Diagram:


Successfully authenticate itself to the server and get access token
Use access token to access server's resources
Intercept username and password
Encrypted Message
Username
Password
JSON web token
Intercept access token
Man-in-the-middle attack
Replay attack
Relay attack

  - ■ Research:
    - ● The current standard for securing data during the transport is TLS or Transport Layer Security protocol. This protocol dictates that on the initial startup of a connection between a client and a server, the two entities will verify each other identity with a trusted third party and share a key that they will use to encrypt and decrypt data during transport. The key is unique for each connection between a client and a server in order to prevent the compromise of past communication in an event that the key gets compromised in the future.
  - ■ Solution:
    - ● The communication channel between a client and the server will be secure with TLS protocol version 1.2 and 1.3.
  - ■ Cost Analysis
    - ● Just like any form of cryptographic-based security, the TLS protocol will introduce an overhead to the system specifically the server and a client will need to encrypt the data before they send it and decrypt data when they receive it. However, the security gains from crippling undetectable passive eavesdroppers outweigh the TLS protocol overhead cost.

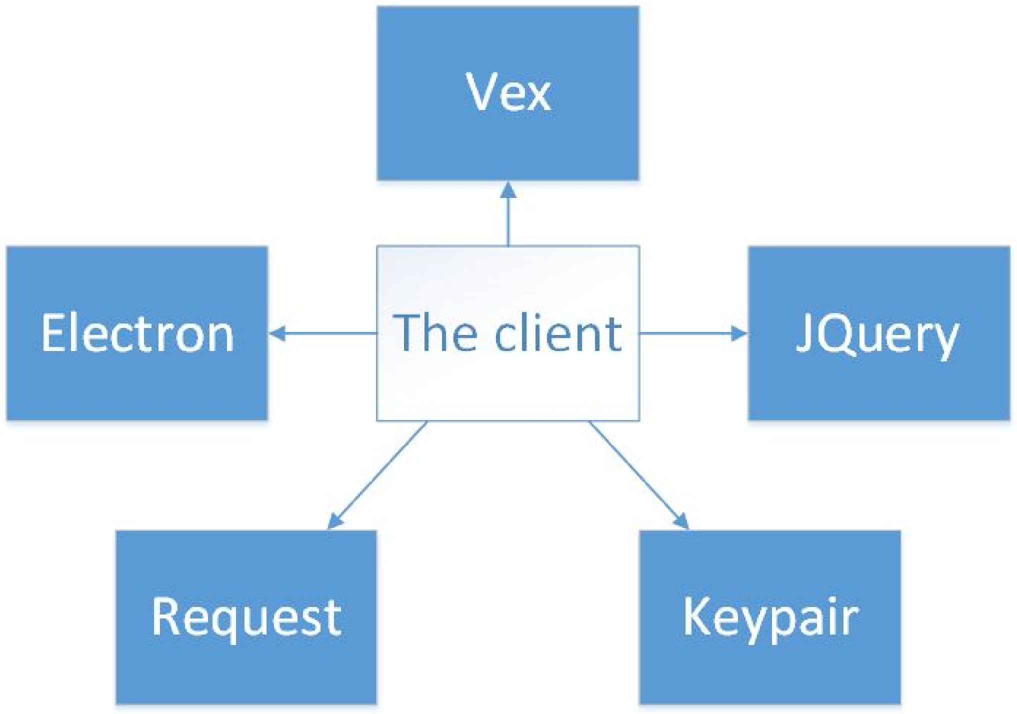
IV. Implementation

Similar to the design phase, each component of the system should be implemented as a standalone application that can function without the need to access resources from other components.

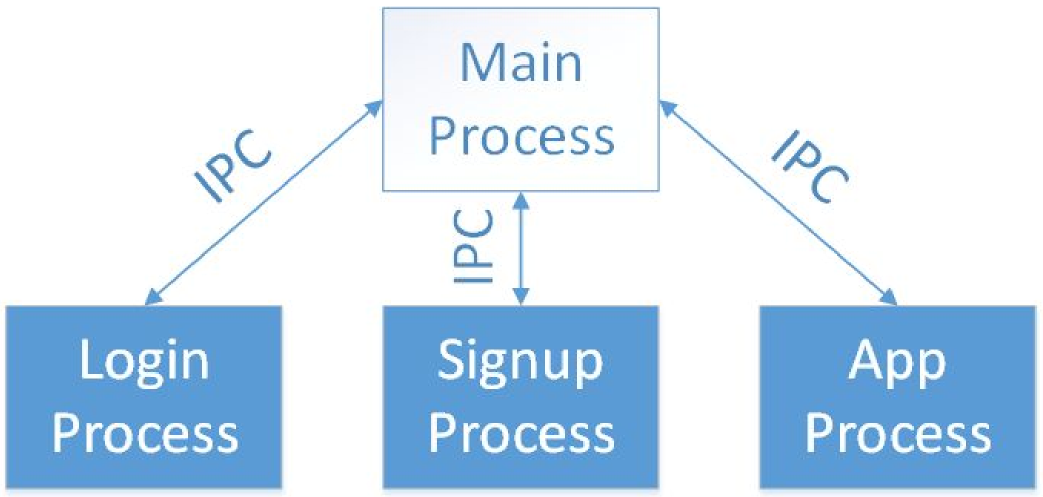
1.Client

The client's responsibilities are an encryption of message before it gets sent to the server and an decryption of messages after it receives from the server and thus, the implementation of the client reflects its responsibilities. In order to smoothen the development process and reduce the cost, the current implementation of the client has a dependency on the following external libraries:

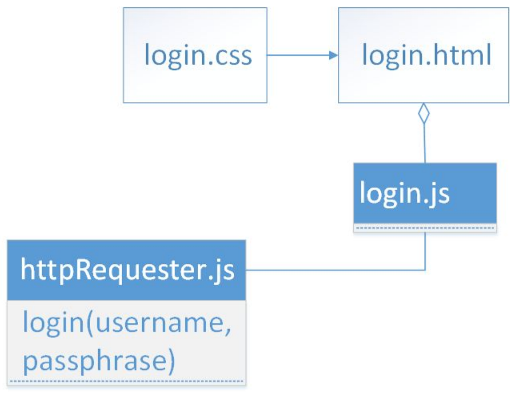
- Electron: a framework allows for the development of desktop GUI application using Node.js for the backend and Chronum for the frontend.
- JQuery: a javascript library that simplifies HTML dom design.
- Keypair: a pure implementation of RSA key pair generation in javascript.
- Request: a javascript library that simplifies the way to make HTTPS call.
- Vex: a library that allows the creation of asynchronous dialogs.



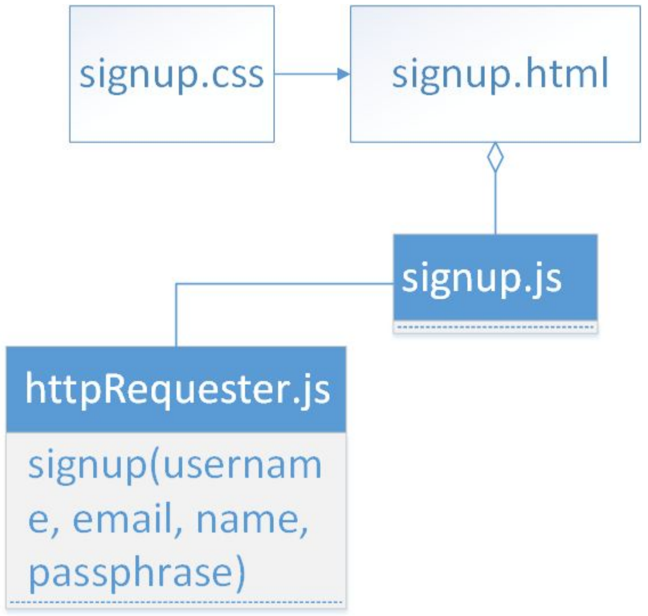
By utilizing the Electron library, the client is structured similarly to an average chromium-based application where there is a centralize main process and multiple rendering processes and the interprocess communication can only occur between the main process and a rendering process. The current implementation of the client has one main process and three rendering processes: login, signup, and app.



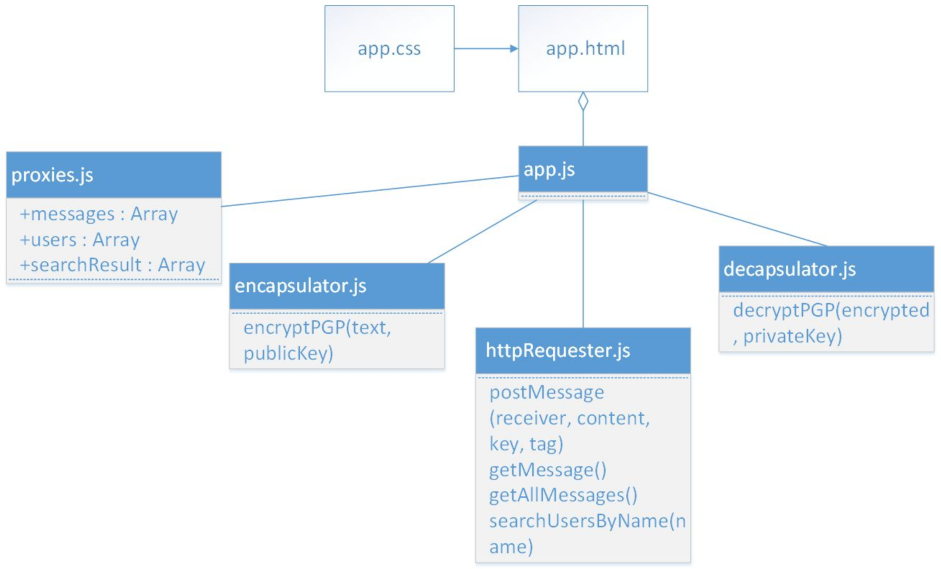
The logic rendering process is responsible for making the HTTPS login call to the server and obtains a JSON web token.



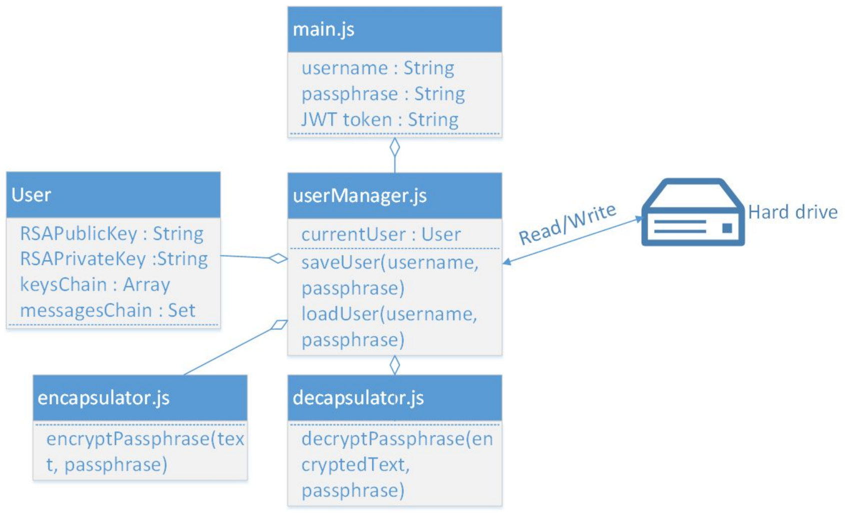
The signup rendering process is responsible for making the HTTPS signup call to the server.



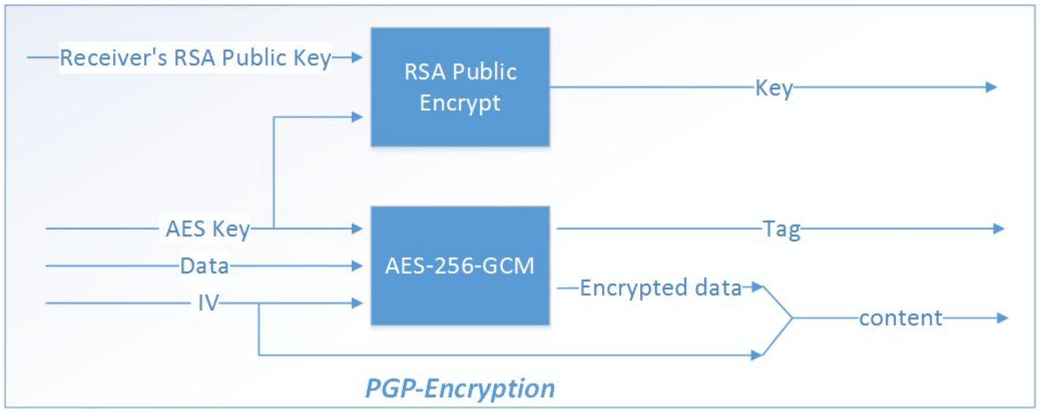
The app rendering process is responsible for sending and receiving messages from the server and the encryption and the decryption of such messages

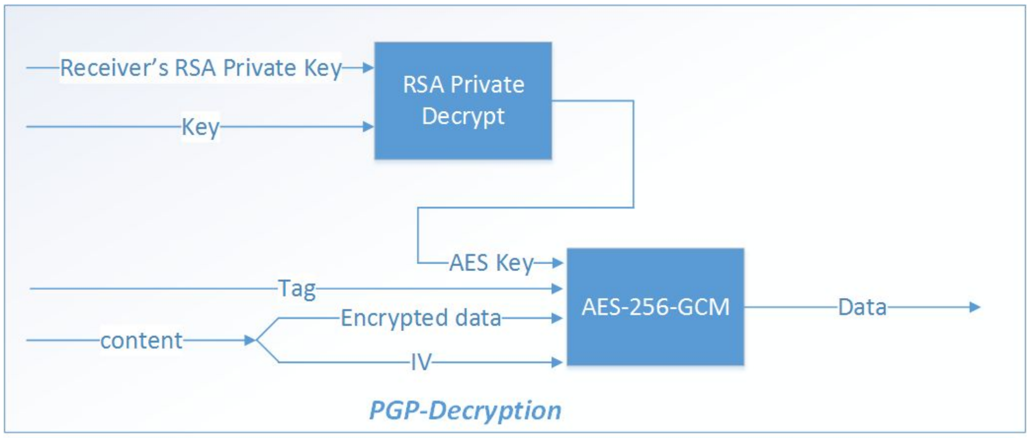


Lastly, the main process is responsible for protecting and managing users' data including their public key, private key and distribute such resources to rendering processes.

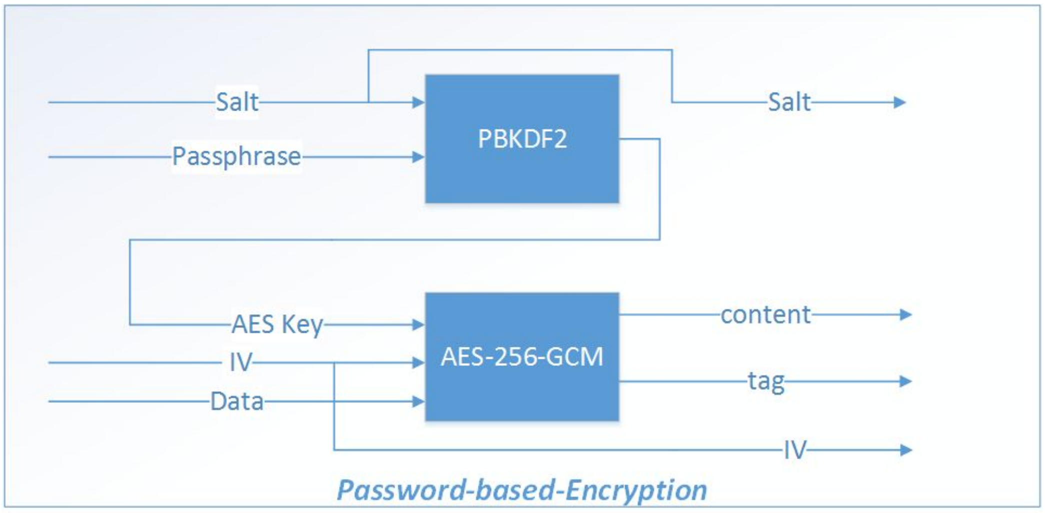


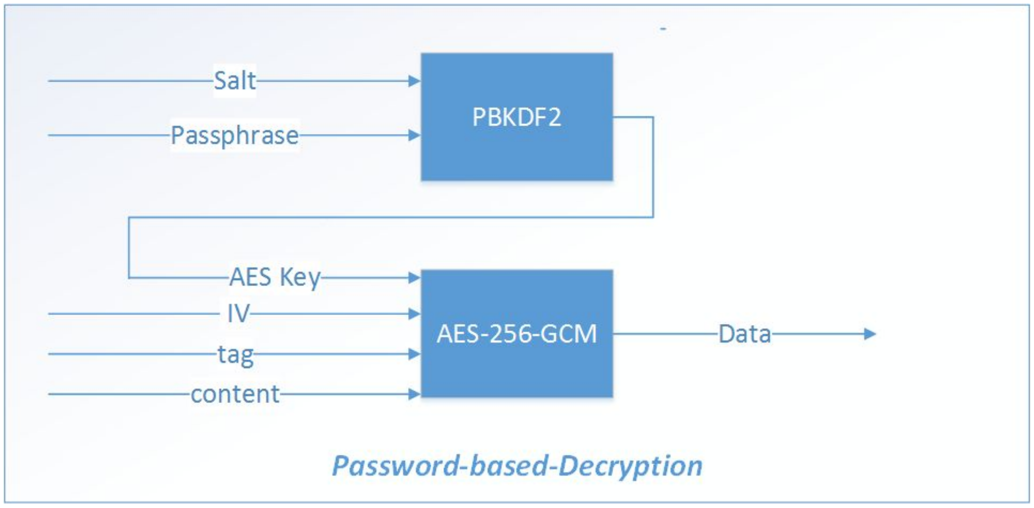
The current implementation of the client utilizes two forms of encryption procedures: PGP and password based. The PGP encryption/decryption is used to protect the message as it is being transported from a client to its recipient.





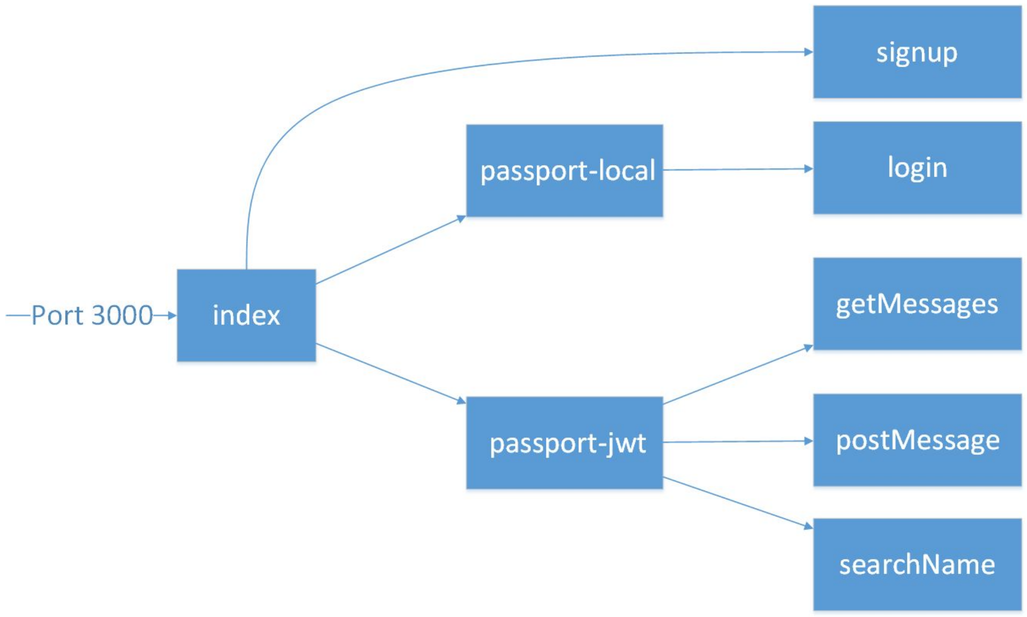
The password-based encryption/decryption is used to protect a user's sensitive information stored in a physical drive.





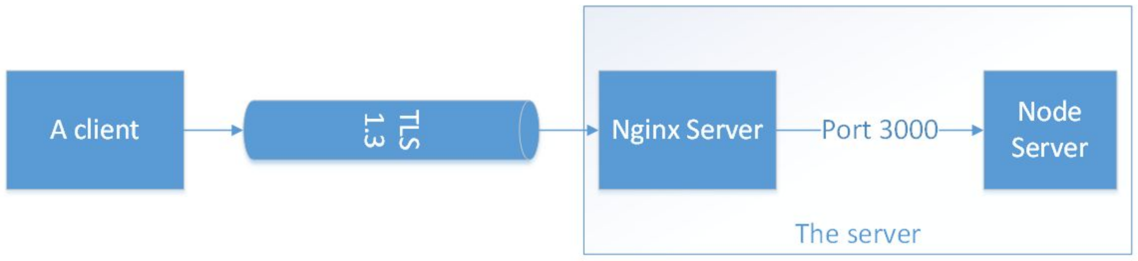
2. Server

The main responsibility of the server is the distribution of messages to clients. In order to fulfill its responsibilities, the server needs to enforce some kind of access control to ensure that a client can only access messages that were sent to it. The current implementation of the server has two categories of routes: unsecured and secured. The unsecured routes include login and signup are ways in which a client can register and authenticate itself to the server. After successfully authenticated itself, the client will receive a temporary JSON web token in which it can be used to access resources in the secured routes.



3. Communication Channel

The communication channel between the server and a client is secured using TLS 1.3 which mean that the data is being encrypted during the transportation. At the initial connection startup, a client and the server will perform a handshake which entails a verification entities' identity with a trusted third party and an exchange of a symmetric key used to encrypt and decrypt data during transport for that session. It is worth noting that the server is a combination of a node server situated before nginx server who acts as the reverse proxy and load balancer.



4.Key Exchange Infrastructure

A major problem of any encryption chat is a method in which users can exchange their public key. On one hand, it is quite convenient for users if the public keys exchange happens automatically with a server acts as a distributor of such keys. On the other hand, the reliant on a server as a public key distributor carries a major risk where the server can perform a man-in-the-middle attack and eavesdropping in clients' conversation.

The current implementation of the system does not offer any official way in which users can exchange their public keys. Thus, it is up users to decide their own out-of-band ways to do the public key exchange such as through email, USB, or letters. Even though this decision will undoubtedly inconvenient users, it also increases the security of the system as each user has a choice to accept only public key of someone he or she already knows.

V. Vulnerabilities and Improvements

No system is perfect and the current implementation of the system is not an exception to this rule. There are some vulnerabilities that exist in the current implementation of the system and such issues should be acknowledged and improve in the future.

To start with, the client is vulnerable to a code injection attack. The reason for this vulnerability is that the client is built using web technologies such as HTML and CSS which is vulnerable to this kind of an attack if the input isn't validated and handle properly. In addition, the vulnerability is exacerbated by the current implementation as it has a complete node integration. This means that an injection code has access to a full-power of Node.js library. Possible fixes to the vulnerability are performing an HTML striping of inputs, turn off HTML parsing of inputs, and reduce the node integration to the only library that the client needs to function.

Similar to the client, the server also has its own set of vulnerabilities. Even though the server has implemented JSON web token to secure its routes, it is still susceptible to DDOS attack as there is no limit of the number of requests a user can make or the signup procedure is too primitive. A remedy to reduce an impact of a DDOS attack is to introduce a request limit so that a user can only make a fixed amount of request per day and to performance email verification of a newly created account to increase the cost of signup procedure. In addition, the server's signature key is semi-static and can only be changed manually. Thus, if the server's signature key gets compromised, there is no way to differentiate between a legitimate token and a fake token. A solution to reduce the problem is to introduce a system that changes the server's signature keys at a fixed interval. The last vulnerability of the server related to the long-term variability of a secure hash function used on the password. There are no viable ways to migrates hash of passwords generated by an obsoleted secure hash function to a new secure hash function.

Last but not least, the communication channel between a client and the server has a long-term viability problem. Currently, a user's password is sent to the server every time he or she attempts to log in and such sensitive data will get encrypted by TLS 1.3. However, every attempt of sending sensitive data is a gamble and it is inevitable that such data will get compromised at some points. Thus, it is important to limit the transport of users' sensitive information. A solution to the above problem is the Remote Login protocol which eliminates the need for users to send their password to the server when they try to log in.

VI. Future Features

- Perform HTML stripping or turn off HTML parsing in a client.
- Reduce node integration in a client.
- Implement a request rate limit in the server.
- Implement an email verification during signup process.
- Implement a signature key rotating system in the server
- Implement remote login protocol.
- Split the server into two independent servers. One server responsible for authenticating users and another server responsible for serving resources.
- Implement key exchange infrastructure.
- Introduce some form of encryption on data stored in the server based on such server's software and hardware.
- Move to symmetric key cryptographic
- Implement forward secrecy
- Add support for group messages and multimedia