Generic Report Template for ECE 572

——————————————————————————

Instructor: Dr. Ardeshir Shojaeinasab

Student Name: Yujian Li

Student ID: V01046555

Assignment: Assignment 2

Date: July 10, 2025

GitHub Repository: https://github.com/YujianLiG208/ECE572

——————————————————————————

Executive Summary

In general, this assignment focuses on three things:

- Implement Time-based One-Time Passwords to strengthen authentication.
- Add third-party authentication using OAuth 2.0 protocols.
- Implement challenge-response protocols and Zero Trust principles.

——————————————————————————

Table of Contents

1. [Introduction]
2. [Task Implementation]

  - [Task 4] Multi-Factor Authentication with TOTP

  - [Task 5] OAuth Integration

  - [Task 6] Zero Trust Implementation

3. [Security Analysis]
4. [Attack Demonstrations]
5. [Performance Evaluation]
6. [Lessons Learned]
7. [Conclusion]
8. [References]

Note: Because copying the code from VSCode into a word document will make it difficult to read and the indentation will be difficult to display properly, all code snippets are screenshots.

——————————————————————————

1. Introduction

1.1 Objective

Explores modern authentication patterns, multi-factor authentication, and Zero Trust security principles.

1.2 Scope

N/A

1.3 Environment Setup

- Operating System: WIN 10, with WSL2 Ubuntu 24.04; KALI Linux Virtual Machine
- Python Version: 3.13.0
- Key Libraries Used: (only new libraries) pyotp, qrcode, qrcode, webbrowser, urllib.parse, requests, logging
- Development Tools: Visual Studio Code with Copilot, VirtualBox, ChatGPT, OAuth, my iphone

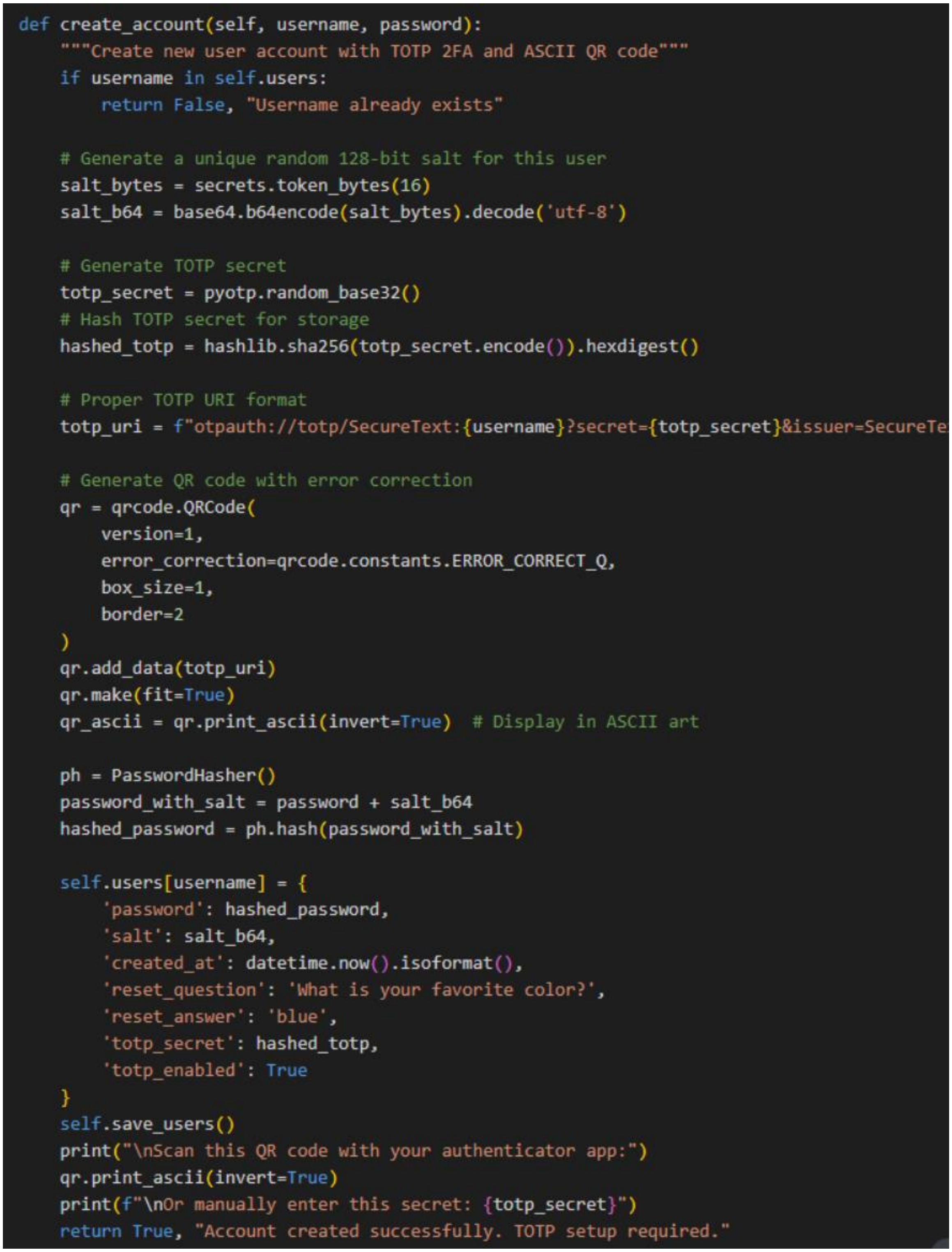
——————————————————————————

2. Task Implementation

2.1 Task 4: Multi-Factor Authentication with TOTP

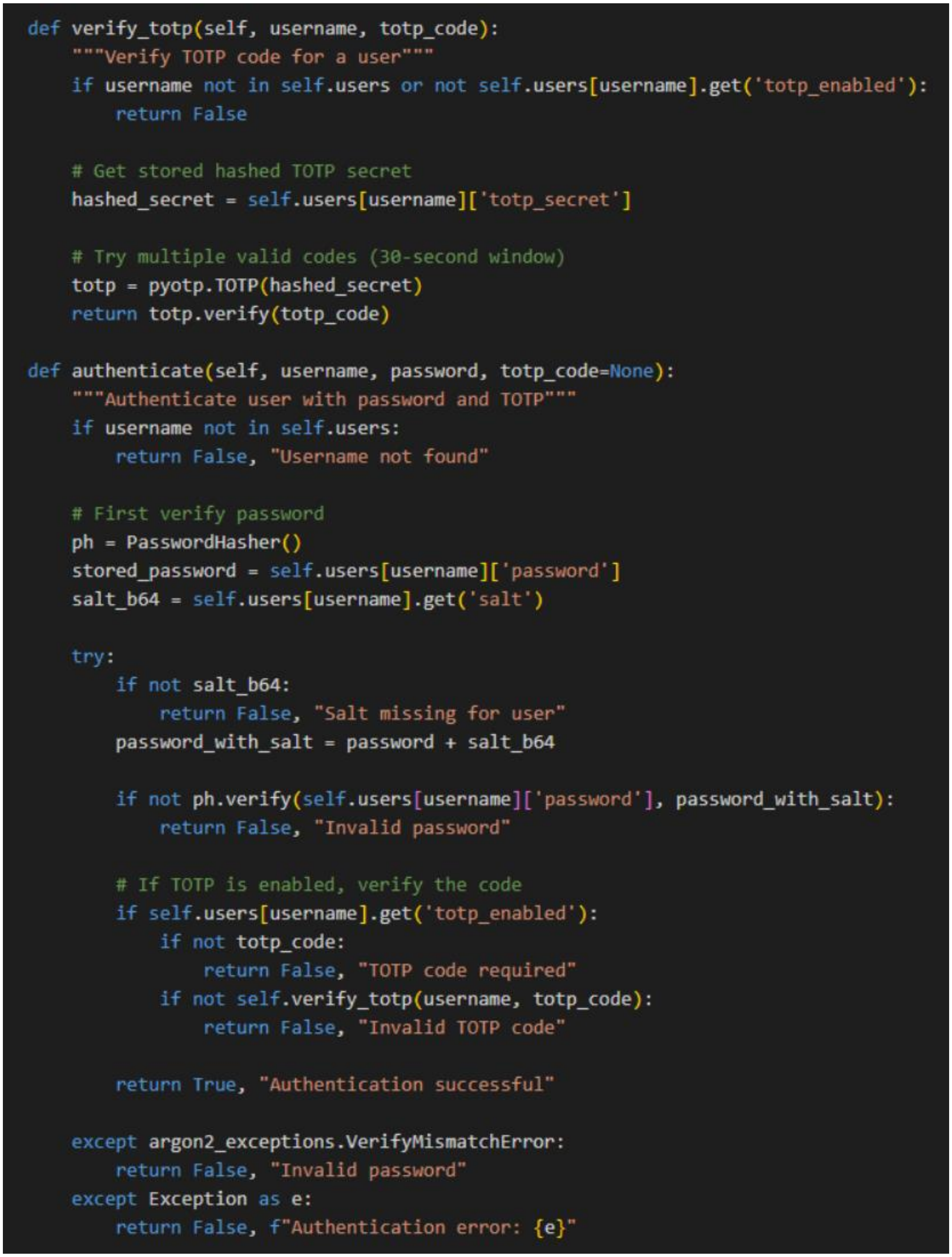
2.1.2 Implementation Details

modify the **create_account** method to generate and store TOTP secrets:



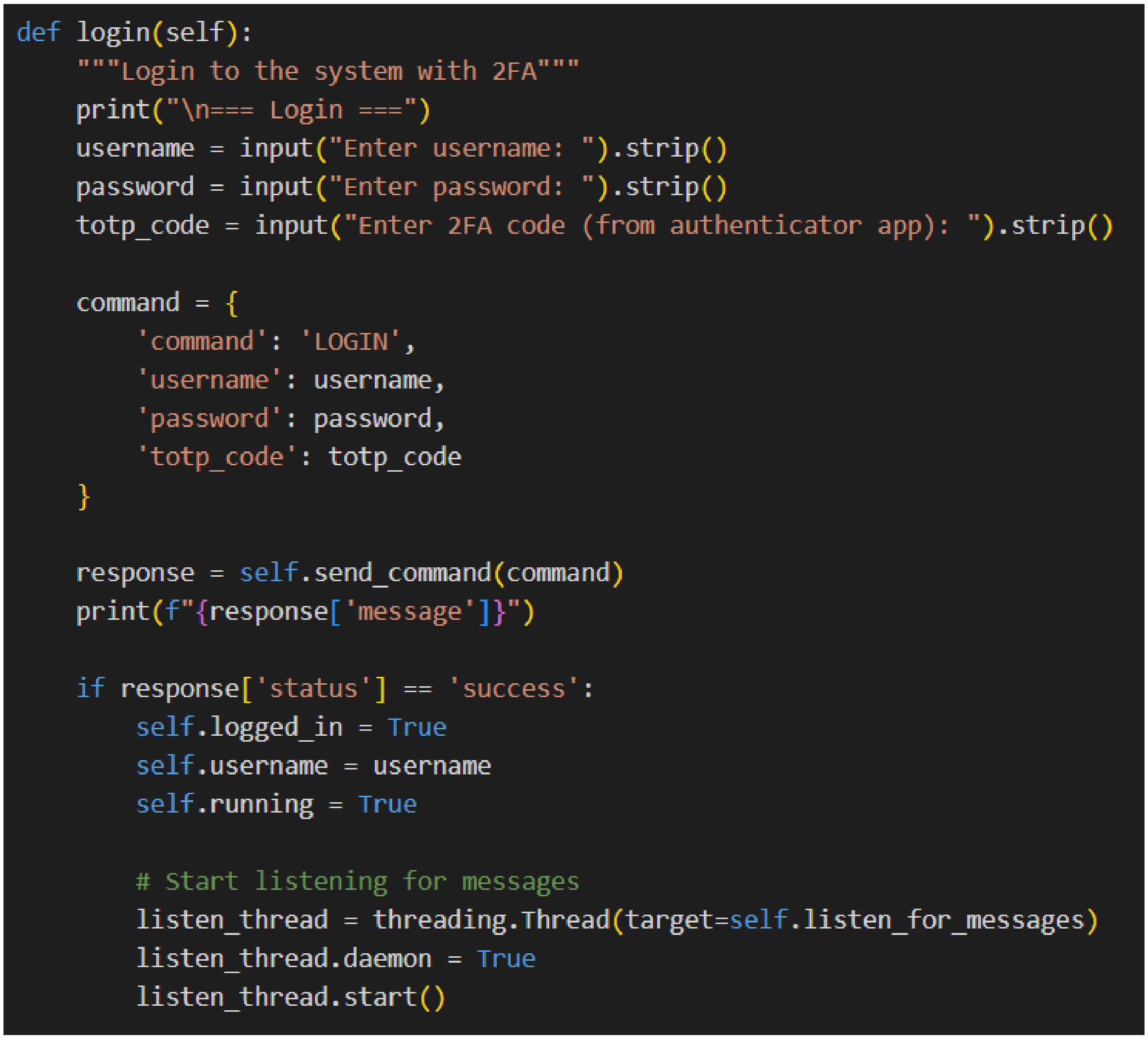
```python
def create_account(self, username, password):
    """Create new user account with TOTP 2FA and ASCII QR code"""
    if username in self.users:
        return False, "Username already exists"

    # Generate a unique random 128-bit salt for this user
    salt_bytes = secrets.token_bytes(16)
    salt_b64 = base64.b64encode(salt_bytes).decode('utf-8')

    # Generate TOTP secret
    totp_secret = pyotp.random_base32()
    # Hash TOTP secret for storage
    hashed_totp = hashlib.sha256(totp_secret.encode()).hexdigest()

    # Proper TOTP URI format
    totp_uri = f"otpauth://totp/SecureText:{username}?secret={totp_secret}&issuer=SecureTe

    # Generate QR code with error correction
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_Q,
        box_size=1,
        border=2
    )
    qr.add_data(totp_uri)
    qr.make(fit=True)
    qr_ascii = qr.print_ascii(invert=True)  # Display in ASCII art

    ph = PasswordHasher()
    password_with_salt = password + salt_b64
    hashed_password = ph.hash(password_with_salt)

    self.users[username] = {
        'password': hashed_password,
        'salt': salt_b64,
        'created_at': datetime.now().isoformat(),
        'reset_question': 'What is your favorite color?',
        'reset_answer': 'blue',
        'totp_secret': hashed_totp,
        'totp_enabled': True
    }
    self.save_users()
    print("\nScan this QR code with your authenticator app:")
    qr.print_ascii(invert=True)
    print(f"\nOr manually enter this secret: {totp_secret}")
    return True, "Account created successfully. TOTP setup required."
```

modify the **authenticate** method to verify TOTP codes:



```
def verify_totp(self, username, totp_code):
    """Verify TOTP code for a user"""
    if username not in self.users or not self.users[username].get('totp_enabled'):
        return False

    # Get stored hashed TOTP secret
    hashed_secret = self.users[username]['totp_secret']

    # Try multiple valid codes (30-second window)
    totp = pyotp.TOTP(hashed_secret)
    return totp.verify(totp_code)

def authenticate(self, username, password, totp_code=None):
    """Authenticate user with password and TOTP"""
    if username not in self.users:
        return False, "Username not found"

    # First verify password
    ph = PasswordHasher()
    stored_password = self.users[username]['password']
    salt_b64 = self.users[username].get('salt')

    try:
        if not salt_b64:
            return False, "Salt missing for user"
        password_with_salt = password + salt_b64

        if not ph.verify(self.users[username]['password'], password_with_salt):
            return False, "Invalid password"

        # If TOTP is enabled, verify the code
        if self.users[username].get('totp_enabled'):
            if not totp_code:
                return False, "TOTP code required"
            if not self.verify_totp(username, totp_code):
                return False, "Invalid TOTP code"

        return True, "Authentication successful"

    except argon2_exceptions.VerifyMismatchError:
        return False, "Invalid password"
    except Exception as e:
        return False, f"Authentication error: {e}"
```

modify the client-side **login** to handle TOTP:

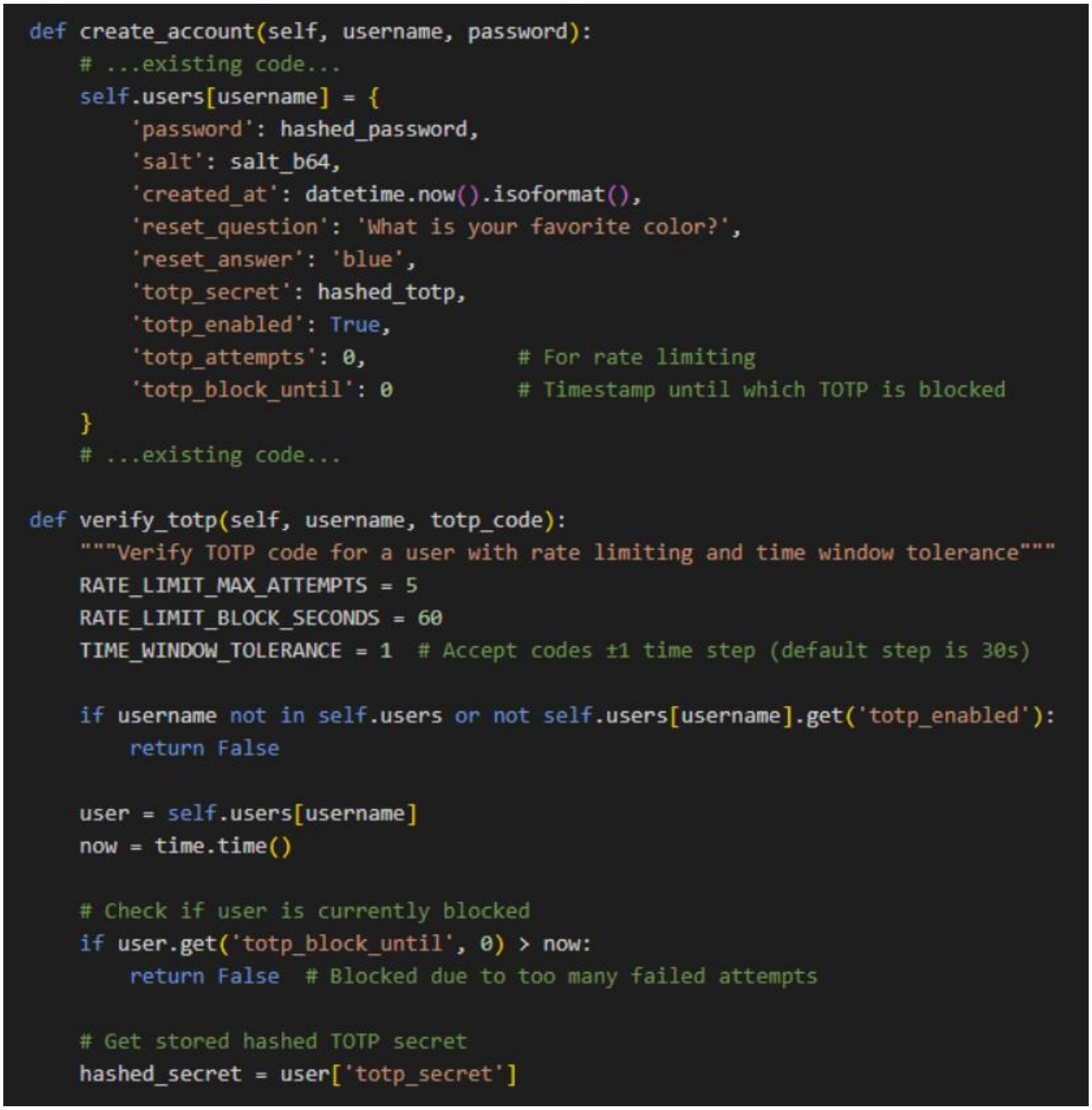


```python
def login(self):
    """Login to the system with 2FA"""
    print("\n=== Login ===")
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()
    totp_code = input("Enter 2FA code (from authenticator app): ").strip()

    command = {
        'command': 'LOGIN',
        'username': username,
        'password': password,
        'totp_code': totp_code
    }

    response = self.send_command(command)
    print(f"{response['message']}")

    if response['status'] == 'success':
        self.logged_in = True
        self.username = username
        self.running = True

        # Start listening for messages
        listen_thread = threading.Thread(target=self.listen_for_messages)
        listen_thread.daemon = True
        listen_thread.start()
```

2.1.3 Challenges and Solutions

Implement rate limiting for TOTP attempts

Add time window tolerance for clock skew

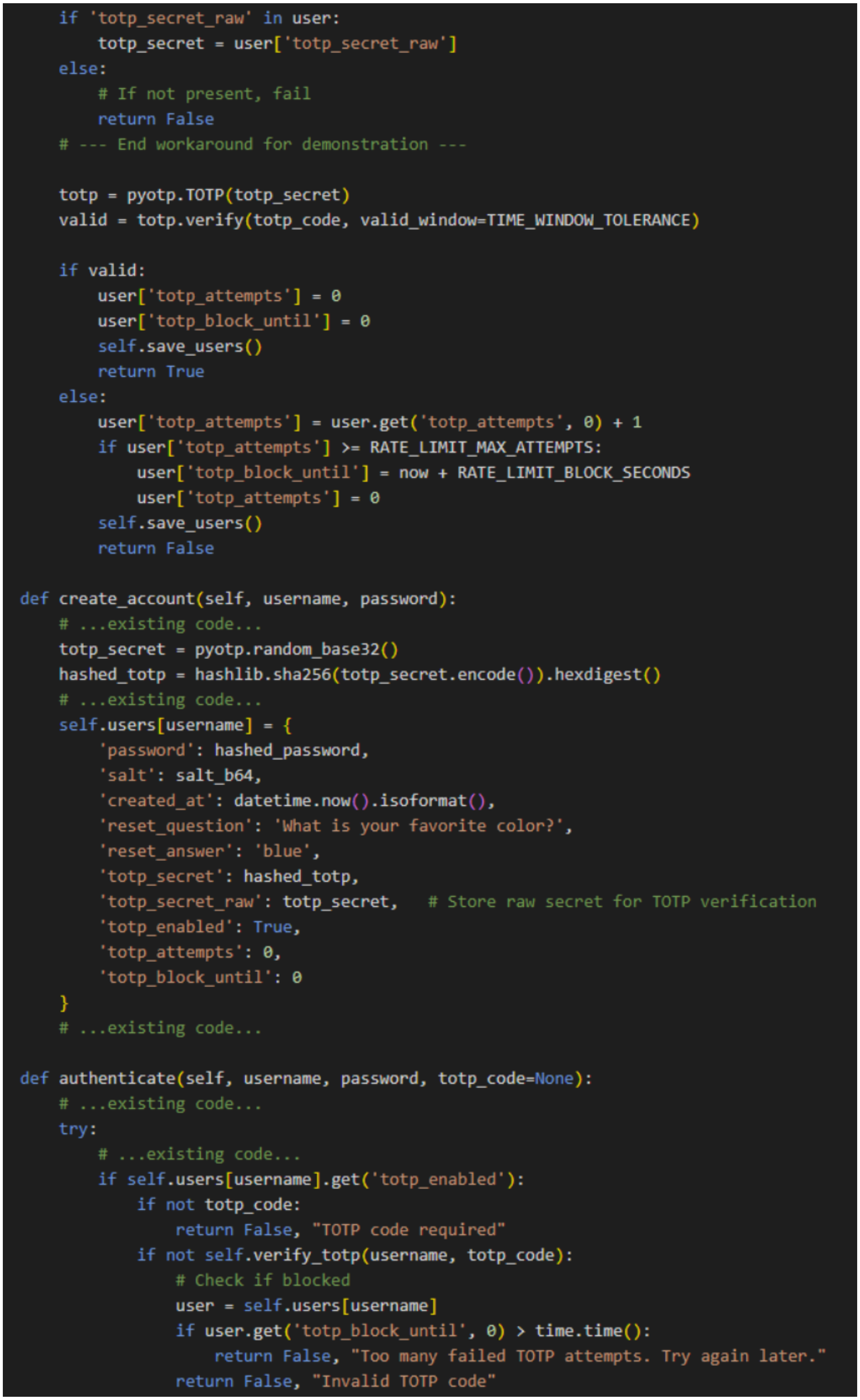


```python
def create_account(self, username, password):
    # ...existing code...
    self.users[username] = {
        'password': hashed_password,
        'salt': salt_b64,
        'created_at': datetime.now().isoformat(),
        'reset_question': 'What is your favorite color?',
        'reset_answer': 'blue',
        'totp_secret': hashed_totp,
        'totp_enabled': True,
        'totp_attempts': 0,           # For rate limiting
        'totp_block_until': 0         # Timestamp until which TOTP is blocked
    }
    # ...existing code...

def verify_totp(self, username, totp_code):
    """Verify TOTP code for a user with rate limiting and time window tolerance"""
    RATE_LIMIT_MAX_ATTEMPTS = 5
    RATE_LIMIT_BLOCK_SECONDS = 60
    TIME_WINDOW_TOLERANCE = 1  # Accept codes ±1 time step (default step is 30s)

    if username not in self.users or not self.users[username].get('totp_enabled'):
        return False

    user = self.users[username]
    now = time.time()

    # Check if user is currently blocked
    if user.get('totp_block_until', 0) > now:
        return False  # Blocked due to too many failed attempts

    # Get stored hashed TOTP secret
    hashed_secret = user['totp_secret']
```

For demonstration, we need the original secret, but only the hash is stored. In a real system, it is required to store the secret securely, not just the hash. Here, we can't verify TOTP with only the hash, so this is a limitation.

let's assume we can retrieve the original secret, totp_secret is the hash, but pyotp needs the original secret. Therefore, we should store the original secret as well (not just the hash).

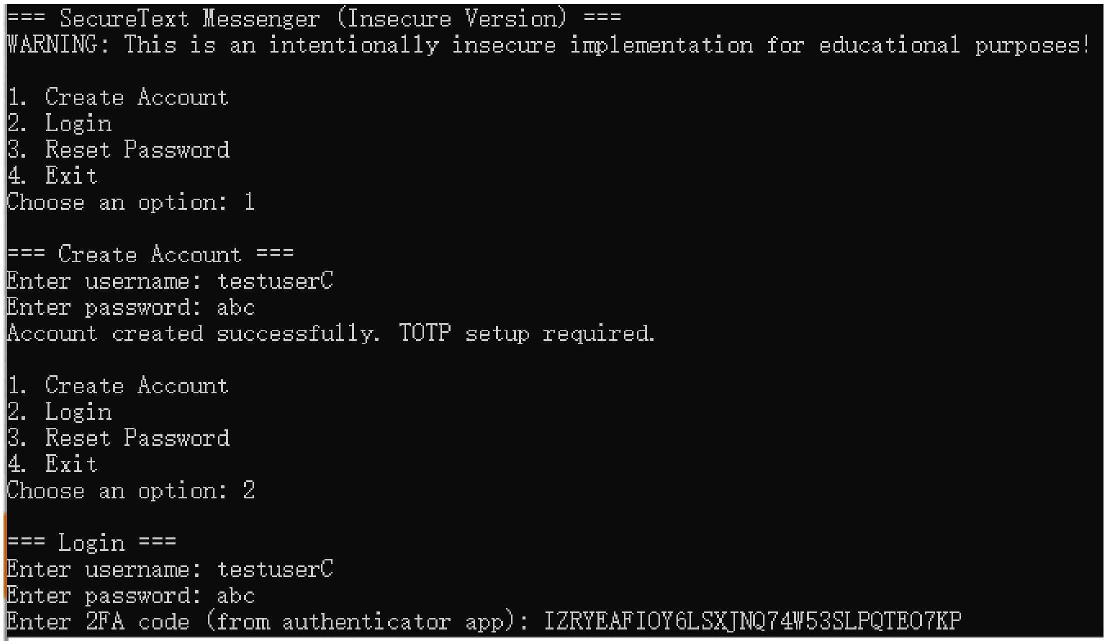
Add 'totp_secret_raw' for this purpose.



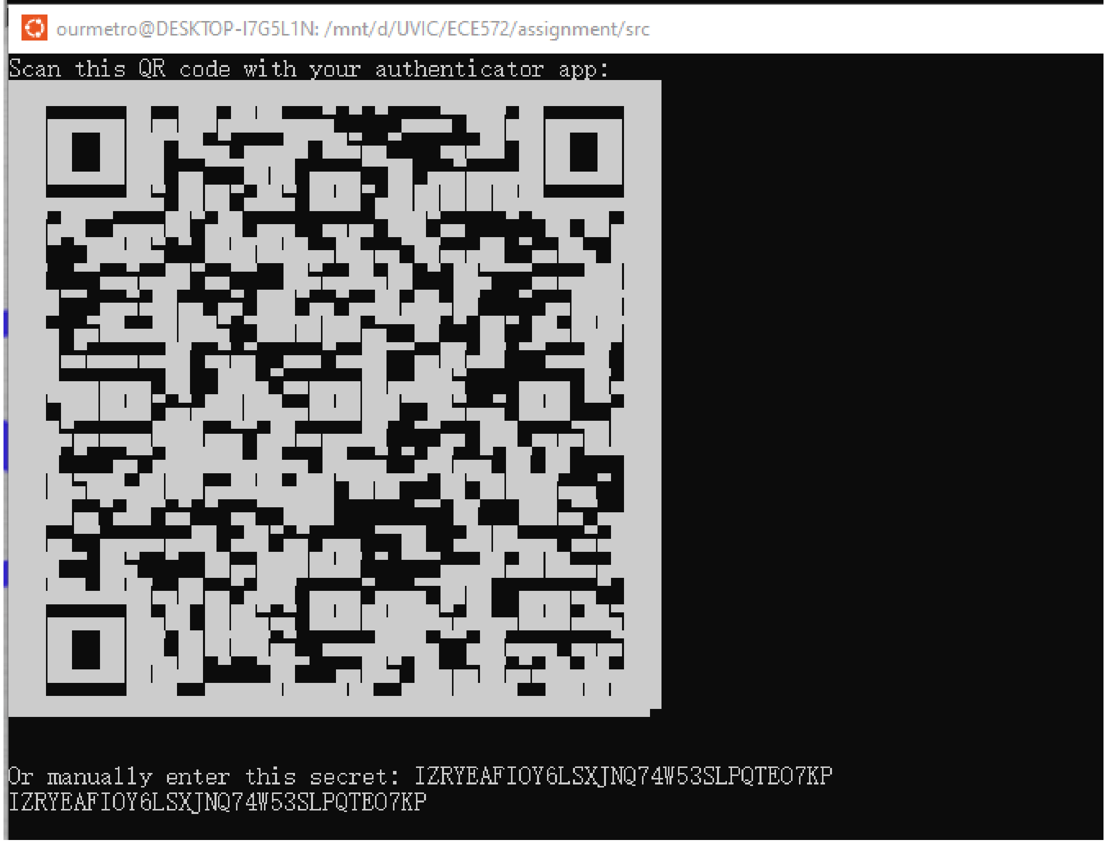
```python
    if 'totp_secret_raw' in user:
        totp_secret = user['totp_secret_raw']
    else:
        # If not present, fail
        return False
    # --- End workaround for demonstration ---

    totp = pyotp.TOTP(totp_secret)
    valid = totp.verify(totp_code, valid_window=TIME_WINDOW_TOLERANCE)

    if valid:
        user['totp_attempts'] = 0
        user['totp_block_until'] = 0
        self.save_users()
        return True
    else:
        user['totp_attempts'] = user.get('totp_attempts', 0) + 1
        if user['totp_attempts'] >= RATE_LIMIT_MAX_ATTEMPTS:
            user['totp_block_until'] = now + RATE_LIMIT_BLOCK_SECONDS
            user['totp_attempts'] = 0
        self.save_users()
        return False

def create_account(self, username, password):
    # ...existing code...
    totp_secret = pyotp.random_base32()
    hashed_totp = hashlib.sha256(totp_secret.encode()).hexdigest()
    # ...existing code...
    self.users[username] = {
        'password': hashed_password,
        'salt': salt_b64,
        'created_at': datetime.now().isoformat(),
        'reset_question': 'What is your favorite color?',
        'reset_answer': 'blue',
        'totp_secret': hashed_totp,
        'totp_secret_raw': totp_secret,   # Store raw secret for TOTP verification
        'totp_enabled': True,
        'totp_attempts': 0,
        'totp_block_until': 0
    }
    # ...existing code...

def authenticate(self, username, password, totp_code=None):
    # ...existing code...
    try:
        # ...existing code...
        if self.users[username].get('totp_enabled'):
            if not totp_code:
                return False, "TOTP code required"
            if not self.verify_totp(username, totp_code):
                # Check if blocked
                user = self.users[username]
                if user.get('totp_block_until', 0) > time.time():
                    return False, "Too many failed TOTP attempts. Try again later."
                return False, "Invalid TOTP code"
```

2.1.4 Testing and Validation

Lets test by trying to create a new account testuserC



```
=== SecureText Messenger (Insecure Version) ===
WARNING: This is an intentionally insecure implementation for educational purposes!

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option: 1

=== Create Account ===
Enter username: testuserC
Enter password: abc
Account created successfully. TOTP setup required.

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option: 2

=== Login ===
Enter username: testuserC
Enter password: abc
Enter 2FA code (from authenticator app): IZRYEAFIOY6LSXJNQ74W53SLPQTEO7KP
```



```
Scan this QR code with your authenticator app:

Or manually enter this secret: IZRYEAFIOY6LSXJNQ74W53SLPQTEO7KP
IZRYEAFIOY6LSXJNQ74W53SLPQTEO7KP
```

QR code is not working at this moment, so I manually entered the secret.

It will work later after we set up the OAuth in task 5.

—————————————————————————

2.2 Task 5: OAuth Integration

2.2.1 Objective

Add third-party authentication using OAuth 2.0 protocols.

(Summarized with ChatGPT)

OAuth 2.0 lets users prove who they are via a trusted identity provider such as GitHub, and then delegate narrowly defined access to countless other apps without ever surrendering their password. Instead of collecting credentials, a relying party receives an access token that is cryptographically bound to the specific scopes the user approved, for example “read-only e-mail” or “post a comment.” This isolation slashes the surface area for password breaches: if the third-party app is hacked, only an easily revocable token leaks, not a reusable master secret. Tokens are deliberately short-lived and can be rotated or revoked on demand, permitting swift incident response without forcing the user to change every stored credential. Because the flow rides over TLS and familiar browser redirects, users encounter the well-known IdP login screen and can layer stronger defenses—hardware security keys, biometrics, risk-based analytics—without the downstream service implementing them. Single Sign-On emerges naturally: one secure session at the IdP unlocks dozens of sites, eliminating the temptation to recycle weak passwords. For developers, OAuth also removes password-storage liabilities, easing compliance.

2.2.2 Implementation Details

Set up OAuth on Github:

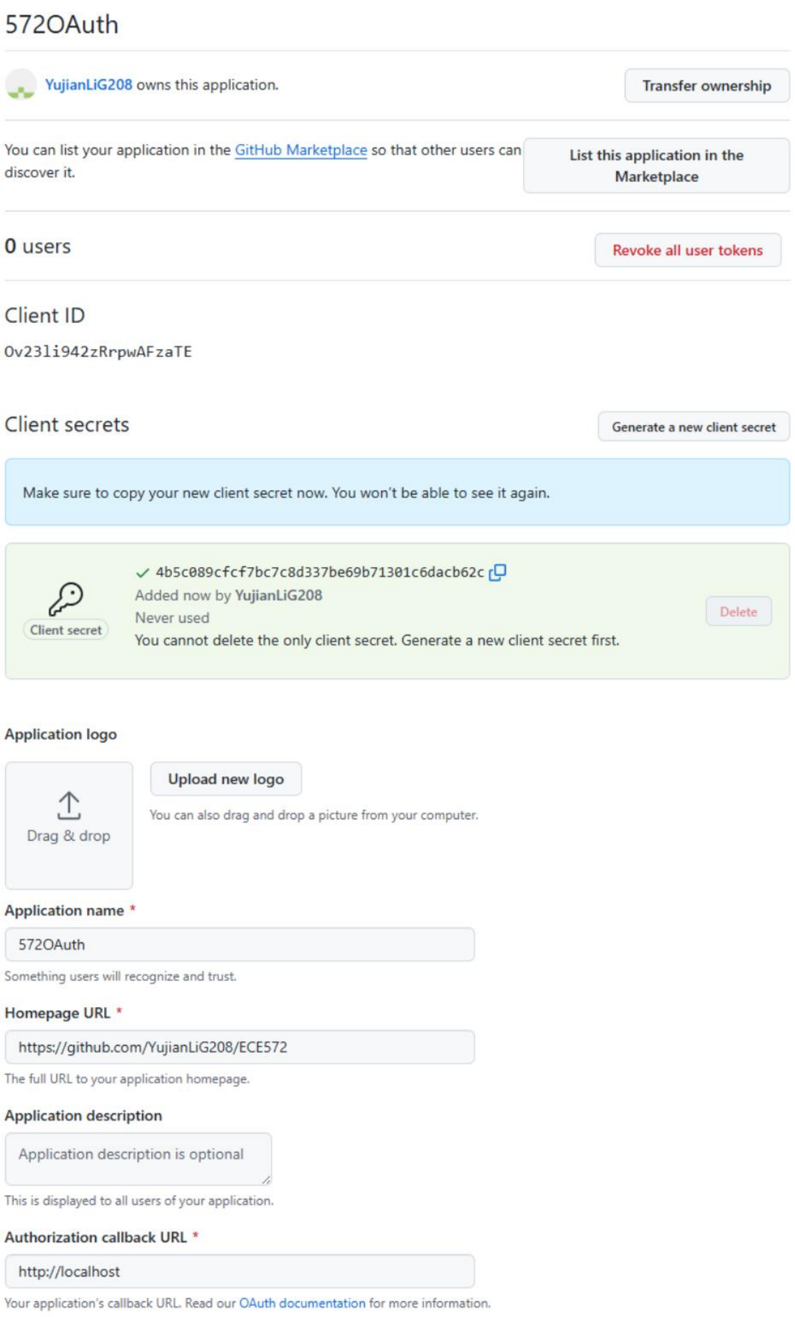
https://github.com/settings/applications/3077790

Client ID

Ov23li942zRrpwAFzaTE

Client secret:

4b5c089cfcf7bc7c8d337be69b71301c6dacb62c



572OAuth

YujianLiG208 owns this application.

You can list your application in the GitHub Marketplace so that other users can discover it.

0 users

Client ID

Ov23li942zRrpwAFzaTE

Client secrets

Make sure to copy your new client secret now. You won't be able to see it again.

✓ 4b5c089cfcf7bc7c8d337be69b71301c6dacb62c
Added now by YujianLiG208
Never used
You cannot delete the only client secret. Generate a new client secret first.

Application logo

You can also drag and drop a picture from your computer.

Application name *

572OAuth

Something users will recognize and trust.

Homepage URL *

https://github.com/YujianLiG208/ECE572

The full URL to your application homepage.

Application description

Application description is optional

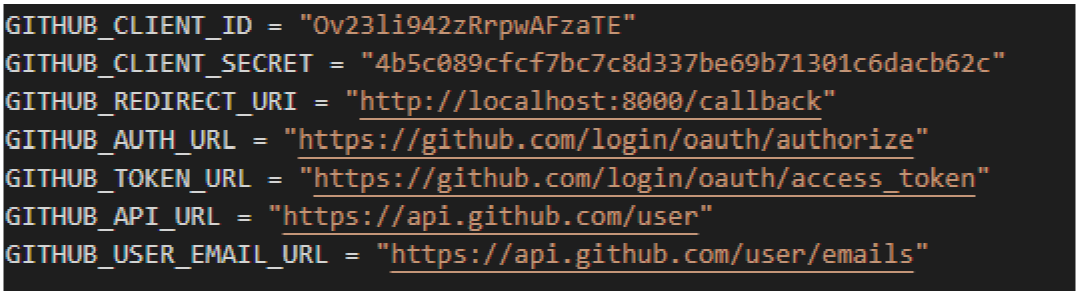
This is displayed to all users of your application.

Authorization callback URL *

http://localhost

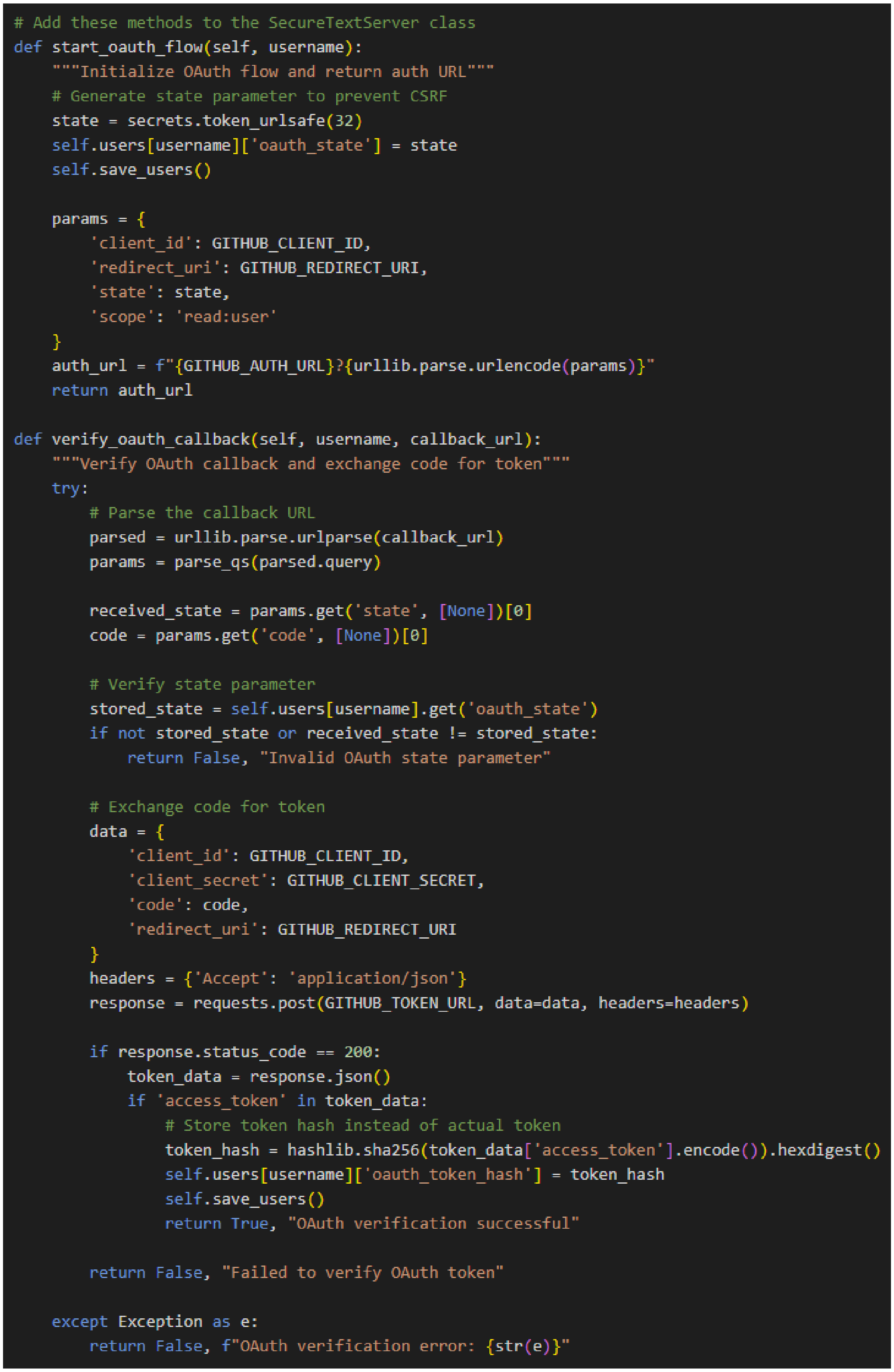
Your application's callback URL. Read our OAuth documentation for more information.

First, I add required imports and OAuth constants and set up the links towards github.



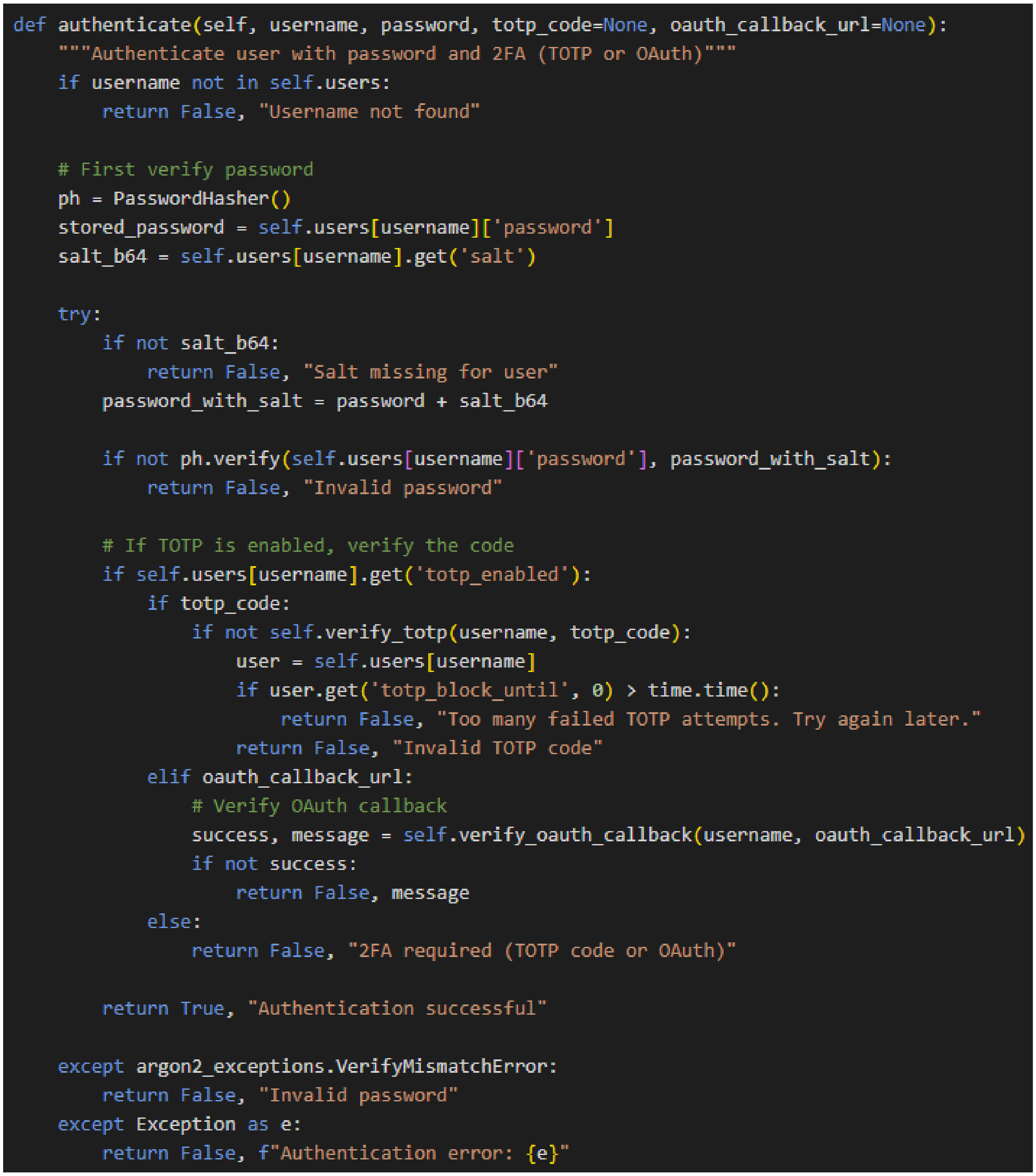
```
GITHUB_CLIENT_ID = "Ov23li942zRrpwAFzaTE"
GITHUB_CLIENT_SECRET = "4b5c089cfcf7bc7c8d337be69b71301c6dacb62c"
GITHUB_REDIRECT_URI = "http://localhost:8000/callback"
GITHUB_AUTH_URL = "https://github.com/login/oauth/authorize"
GITHUB_TOKEN_URL = "https://github.com/login/oauth/access_token"
GITHUB_API_URL = "https://api.github.com/user"
GITHUB_USER_EMAIL_URL = "https://api.github.com/user/emails"
```

Add OAuth methods to the SecureTextServer class:



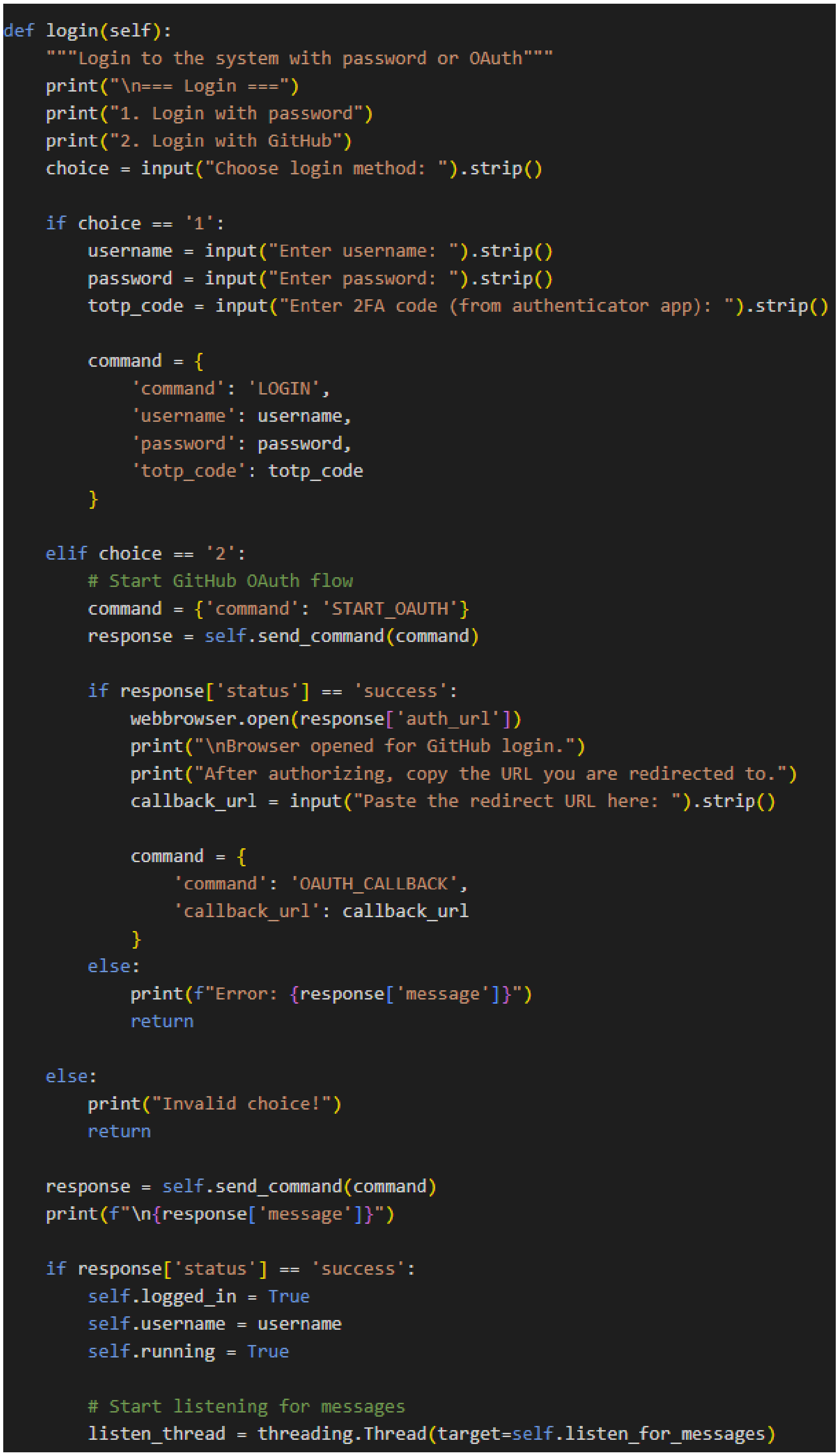
```python
# Add these methods to the SecureTextServer class
def start_oauth_flow(self, username):
    """Initialize OAuth flow and return auth URL"""
    # Generate state parameter to prevent CSRF
    state = secrets.token_urlsafe(32)
    self.users[username]['oauth_state'] = state
    self.save_users()

    params = {
        'client_id': GITHUB_CLIENT_ID,
        'redirect_uri': GITHUB_REDIRECT_URI,
        'state': state,
        'scope': 'read:user'
    }
    auth_url = f"{GITHUB_AUTH_URL}?{urllib.parse.urlencode(params)}"
    return auth_url

def verify_oauth_callback(self, username, callback_url):
    """Verify OAuth callback and exchange code for token"""
    try:
        # Parse the callback URL
        parsed = urllib.parse.urlparse(callback_url)
        params = parse_qs(parsed.query)

        received_state = params.get('state', [None])[0]
        code = params.get('code', [None])[0]

        # Verify state parameter
        stored_state = self.users[username].get('oauth_state')
        if not stored_state or received_state != stored_state:
            return False, "Invalid OAuth state parameter"

        # Exchange code for token
        data = {
            'client_id': GITHUB_CLIENT_ID,
            'client_secret': GITHUB_CLIENT_SECRET,
            'code': code,
            'redirect_uri': GITHUB_REDIRECT_URI
        }
        headers = {'Accept': 'application/json'}
        response = requests.post(GITHUB_TOKEN_URL, data=data, headers=headers)

        if response.status_code == 200:
            token_data = response.json()
            if 'access_token' in token_data:
                # Store token hash instead of actual token
                token_hash = hashlib.sha256(token_data['access_token'].encode()).hexdigest()
                self.users[username]['oauth_token_hash'] = token_hash
                self.save_users()
                return True, "OAuth verification successful"

        return False, "Failed to verify OAuth token"

    except Exception as e:
        return False, f"OAuth verification error: {str(e)}"
```

Modify the authenticate method to support OAuth:



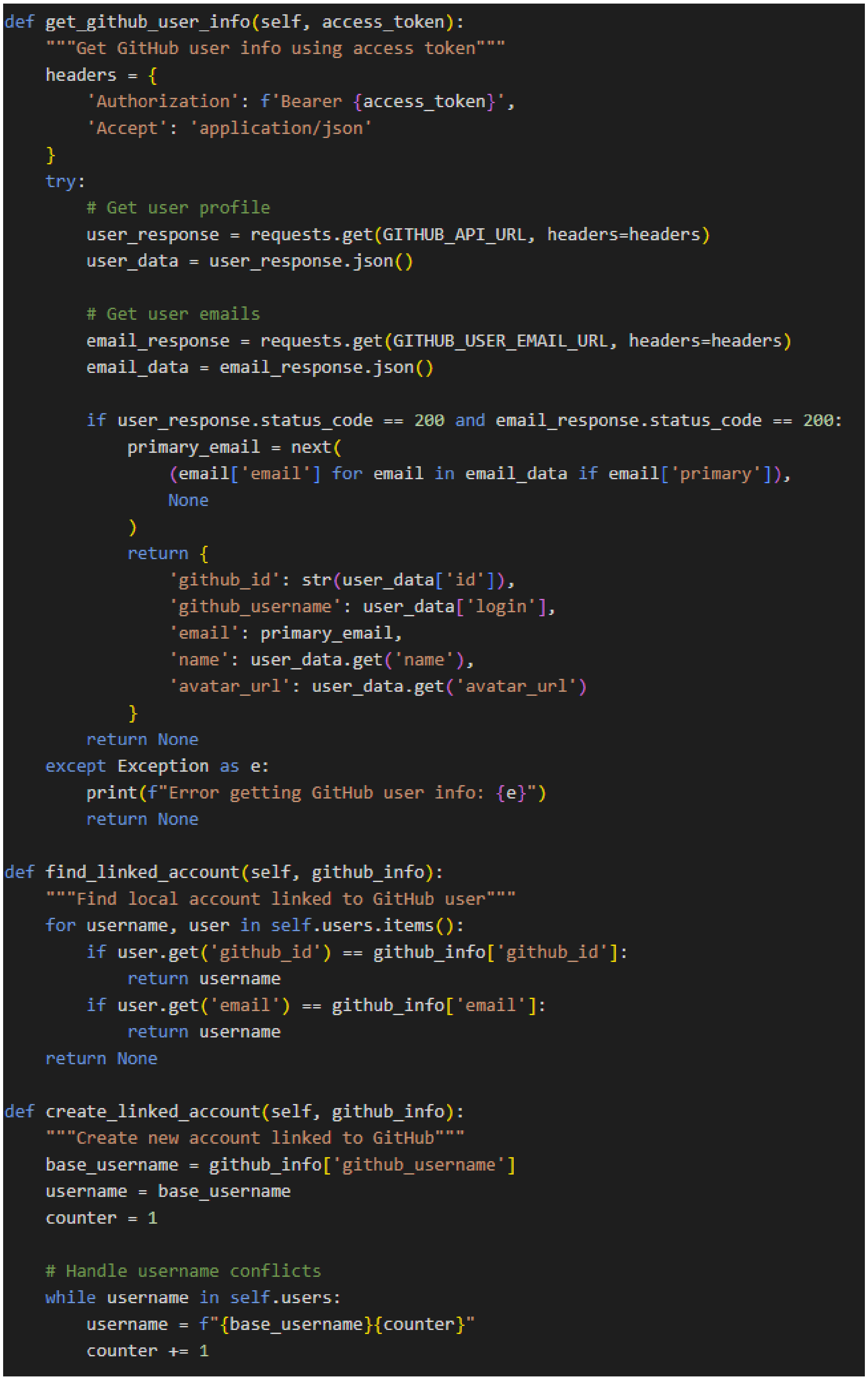
```python
def authenticate(self, username, password, totp_code=None, oauth_callback_url=None):
    """Authenticate user with password and 2FA (TOTP or OAuth)"""
    if username not in self.users:
        return False, "Username not found"

    # First verify password
    ph = PasswordHasher()
    stored_password = self.users[username]['password']
    salt_b64 = self.users[username].get('salt')

    try:
        if not salt_b64:
            return False, "Salt missing for user"
        password_with_salt = password + salt_b64

        if not ph.verify(self.users[username]['password'], password_with_salt):
            return False, "Invalid password"

        # If TOTP is enabled, verify the code
        if self.users[username].get('totp_enabled'):
            if totp_code:
                if not self.verify_totp(username, totp_code):
                    user = self.users[username]
                    if user.get('totp_block_until', 0) > time.time():
                        return False, "Too many failed TOTP attempts. Try again later."
                    return False, "Invalid TOTP code"
            elif oauth_callback_url:
                # Verify OAuth callback
                success, message = self.verify_oauth_callback(username, oauth_callback_url)
                if not success:
                    return False, message
            else:
                return False, "2FA required (TOTP code or OAuth)"

        return True, "Authentication successful"

    except argon2_exceptions.VerifyMismatchError:
        return False, "Invalid password"
    except Exception as e:
        return False, f"Authentication error: {e}"
```

Modify the client login method



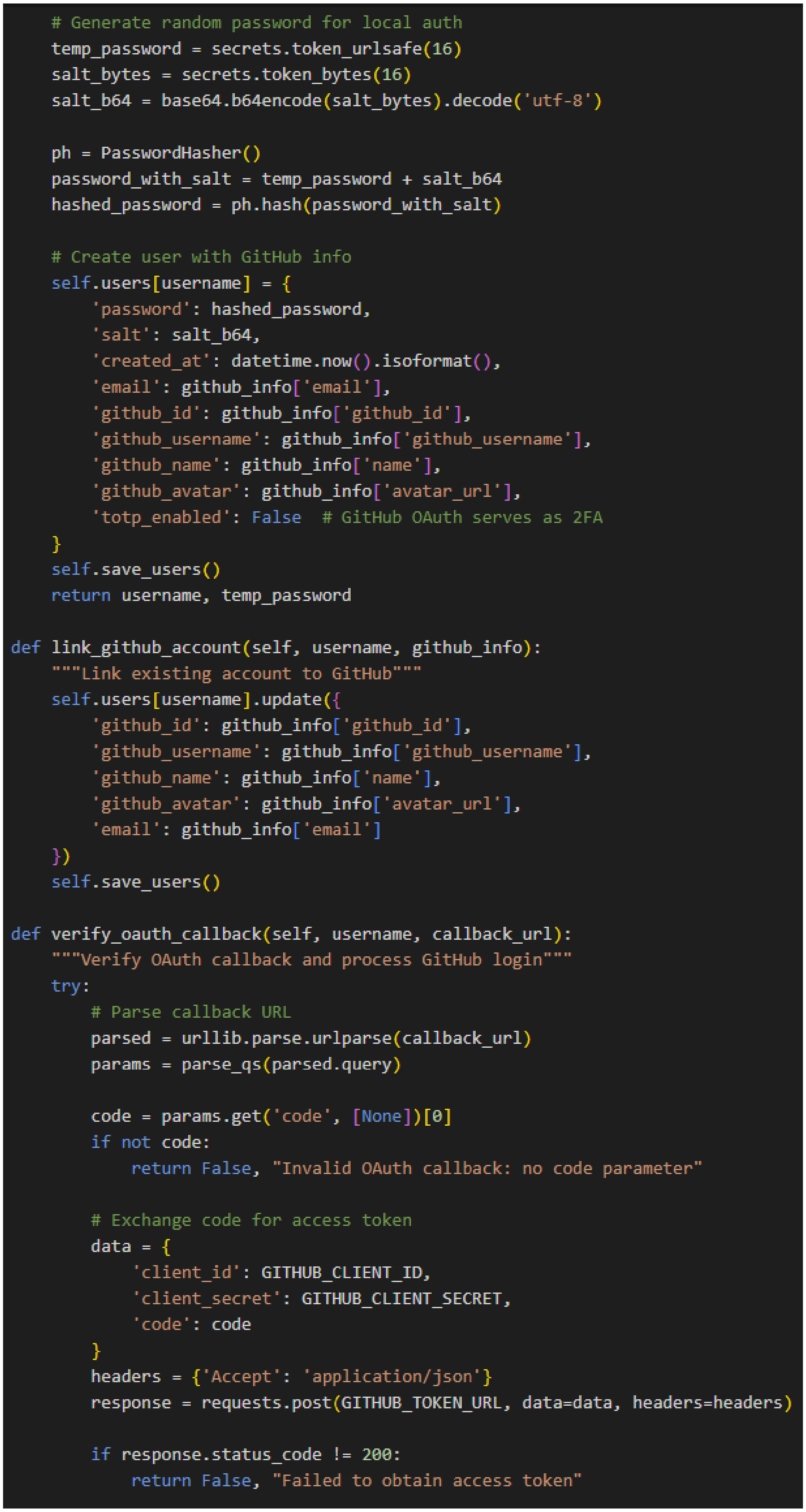
```python
def login(self):
    """Login to the system with password or OAuth"""
    print("\n=== Login ===")
    print("1. Login with password")
    print("2. Login with GitHub")
    choice = input("Choose login method: ").strip()

    if choice == '1':
        username = input("Enter username: ").strip()
        password = input("Enter password: ").strip()
        totp_code = input("Enter 2FA code (from authenticator app): ").strip()

        command = {
            'command': 'LOGIN',
            'username': username,
            'password': password,
            'totp_code': totp_code
        }

    elif choice == '2':
        # Start GitHub OAuth flow
        command = {'command': 'START_OAUTH'}
        response = self.send_command(command)

        if response['status'] == 'success':
            webbrowser.open(response['auth_url'])
            print("\nBrowser opened for GitHub login.")
            print("After authorizing, copy the URL you are redirected to.")
            callback_url = input("Paste the redirect URL here: ").strip()

            command = {
                'command': 'OAUTH_CALLBACK',
                'callback_url': callback_url
            }
        else:
            print(f"Error: {response['message']}")
            return

    else:
        print("Invalid choice!")
        return

    response = self.send_command(command)
    print(f"\n{response['message']}")

    if response['status'] == 'success':
        self.logged_in = True
        self.username = username
        self.running = True

        # Start listening for messages
        listen_thread = threading.Thread(target=self.listen_for_messages)
```

New method to allow github account login

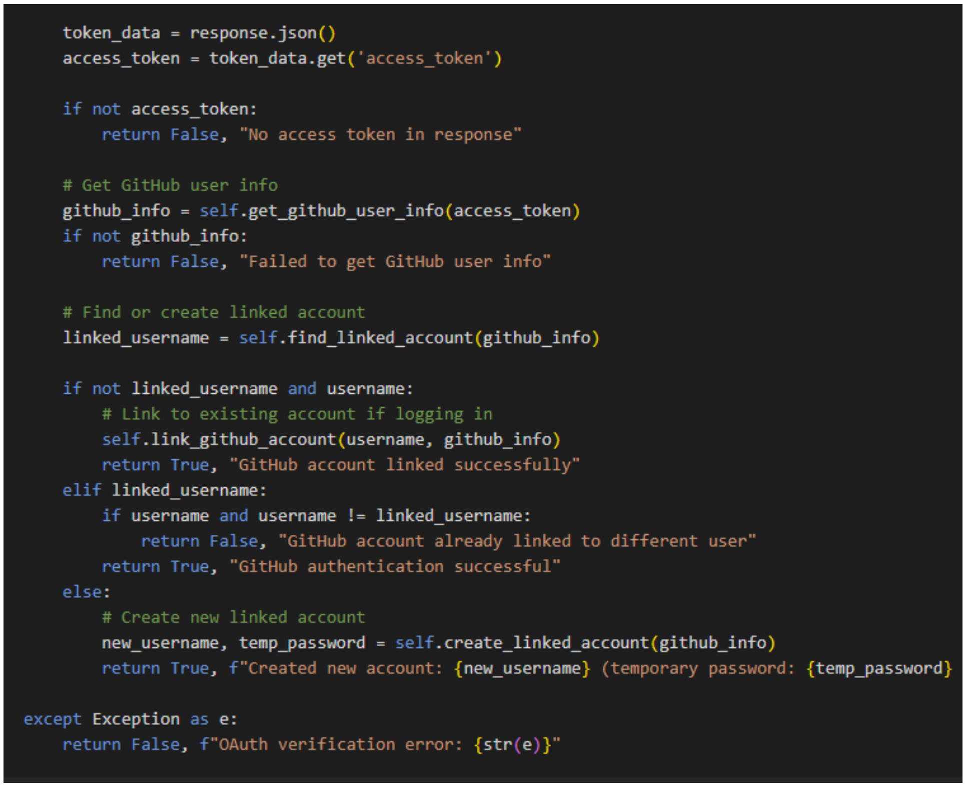


```python
def get_github_user_info(self, access_token):
    """Get GitHub user info using access token"""
    headers = {
        'Authorization': f'Bearer {access_token}',
        'Accept': 'application/json'
    }
    try:
        # Get user profile
        user_response = requests.get(GITHUB_API_URL, headers=headers)
        user_data = user_response.json()

        # Get user emails
        email_response = requests.get(GITHUB_USER_EMAIL_URL, headers=headers)
        email_data = email_response.json()

        if user_response.status_code == 200 and email_response.status_code == 200:
            primary_email = next(
                (email['email'] for email in email_data if email['primary']),
                None
            )
            return {
                'github_id': str(user_data['id']),
                'github_username': user_data['login'],
                'email': primary_email,
                'name': user_data.get('name'),
                'avatar_url': user_data.get('avatar_url')
            }
        return None
    except Exception as e:
        print(f"Error getting GitHub user info: {e}")
        return None

def find_linked_account(self, github_info):
    """Find local account linked to GitHub user"""
    for username, user in self.users.items():
        if user.get('github_id') == github_info['github_id']:
            return username
        if user.get('email') == github_info['email']:
            return username
    return None

def create_linked_account(self, github_info):
    """Create new account linked to GitHub"""
    base_username = github_info['github_username']
    username = base_username
    counter = 1

    # Handle username conflicts
    while username in self.users:
        username = f"{base_username}{counter}"
        counter += 1
```

(contiunes)



```
    # Generate random password for local auth
    temp_password = secrets.token_urlsafe(16)
    salt_bytes = secrets.token_bytes(16)
    salt_b64 = base64.b64encode(salt_bytes).decode('utf-8')

    ph = PasswordHasher()
    password_with_salt = temp_password + salt_b64
    hashed_password = ph.hash(password_with_salt)

    # Create user with GitHub info
    self.users[username] = {
        'password': hashed_password,
        'salt': salt_b64,
        'created_at': datetime.now().isoformat(),
        'email': github_info['email'],
        'github_id': github_info['github_id'],
        'github_username': github_info['github_username'],
        'github_name': github_info['name'],
        'github_avatar': github_info['avatar_url'],
        'totp_enabled': False  # GitHub OAuth serves as 2FA
    }
    self.save_users()
    return username, temp_password

def link_github_account(self, username, github_info):
    """Link existing account to GitHub"""
    self.users[username].update({
        'github_id': github_info['github_id'],
        'github_username': github_info['github_username'],
        'github_name': github_info['name'],
        'github_avatar': github_info['avatar_url'],
        'email': github_info['email']
    })
    self.save_users()

def verify_oauth_callback(self, username, callback_url):
    """Verify OAuth callback and process GitHub login"""
    try:
        # Parse callback URL
        parsed = urllib.parse.urlparse(callback_url)
        params = parse_qs(parsed.query)

        code = params.get('code', [None])[0]
        if not code:
            return False, "Invalid OAuth callback: no code parameter"

        # Exchange code for access token
        data = {
            'client_id': GITHUB_CLIENT_ID,
            'client_secret': GITHUB_CLIENT_SECRET,
            'code': code
        }
        headers = {'Accept': 'application/json'}
        response = requests.post(GITHUB_TOKEN_URL, data=data, headers=headers)

        if response.status_code != 200:
            return False, "Failed to obtain access token"
```

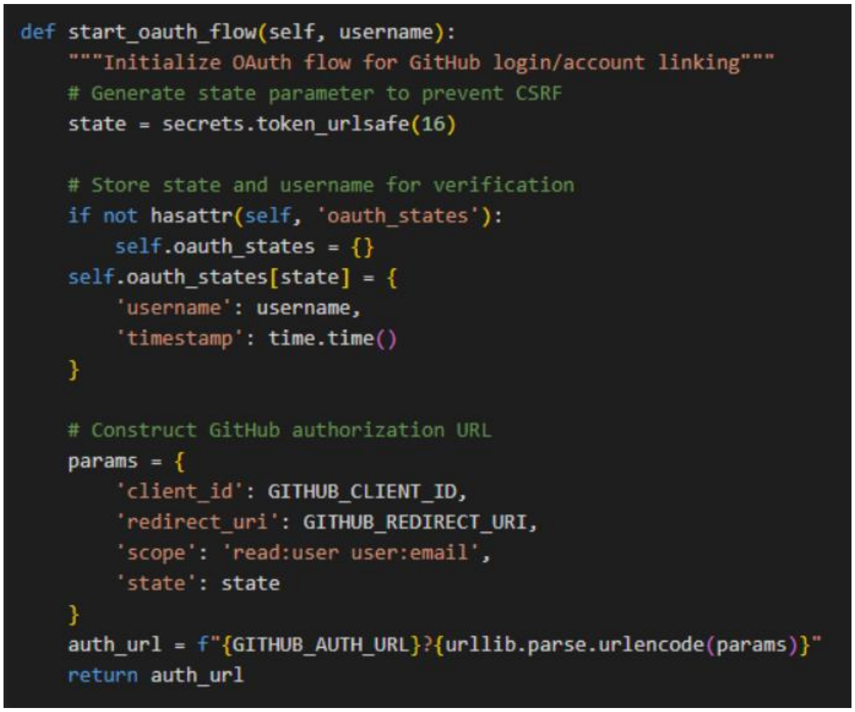


```
    token_data = response.json()
    access_token = token_data.get('access_token')

    if not access_token:
        return False, "No access token in response"

    # Get GitHub user info
    github_info = self.get_github_user_info(access_token)
    if not github_info:
        return False, "Failed to get GitHub user info"

    # Find or create linked account
    linked_username = self.find_linked_account(github_info)

    if not linked_username and username:
        # Link to existing account if logging in
        self.link_github_account(username, github_info)
        return True, "GitHub account linked successfully"
    elif linked_username:
        if username and username != linked_username:
            return False, "GitHub account already linked to different user"
        return True, "GitHub authentication successful"
    else:
        # Create new linked account
        new_username, temp_password = self.create_linked_account(github_info)
        return True, f"Created new account: {new_username} (temporary password: {temp_password}

except Exception as e:
    return False, f"OAuth verification error: {str(e)}"
```

2.2.3 Challenges and Solutions

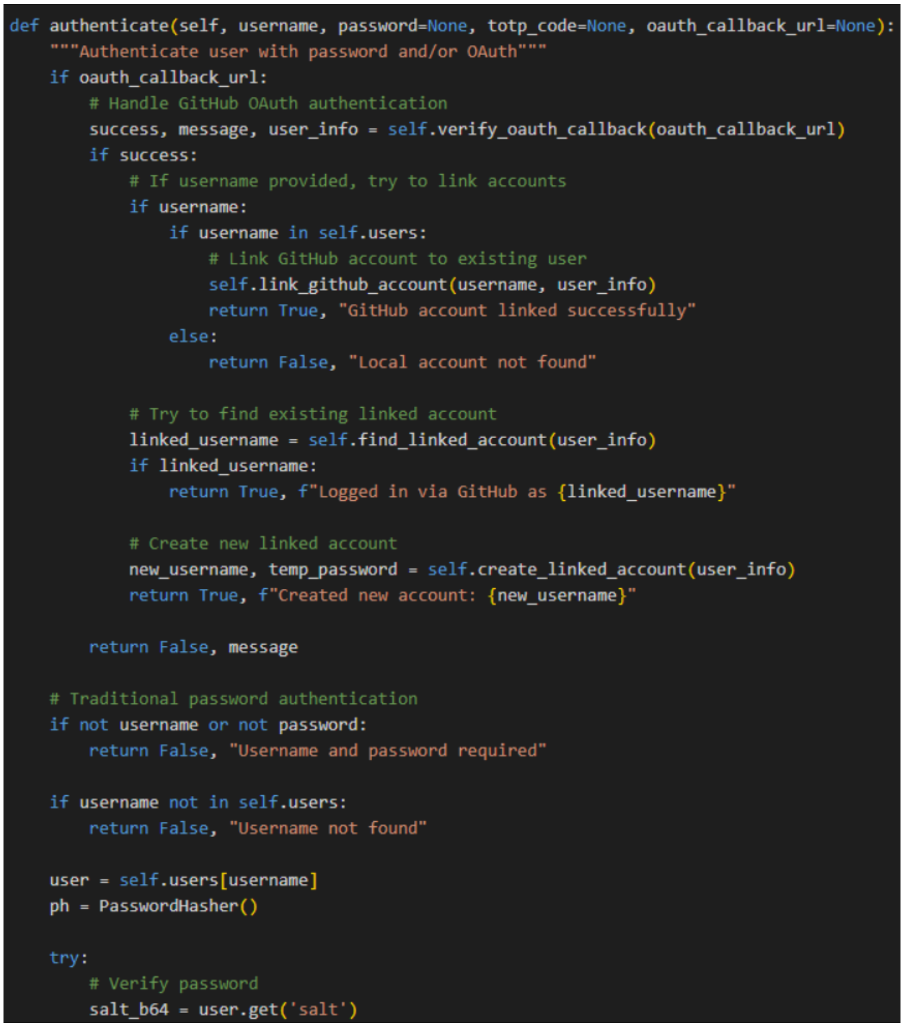
Imply hybrid authentication to let securetext support login via both Local accounts & GitHub OAuth

add a method to handle the OAuth flow initialization:

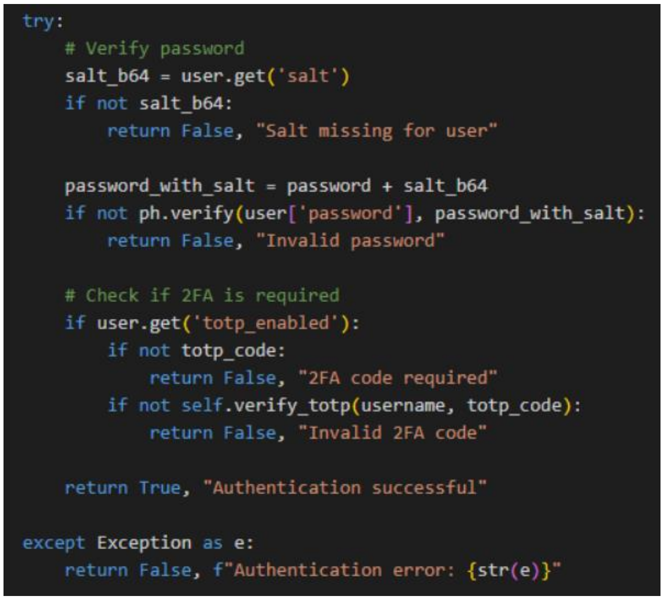


```
def start_oauth_flow(self, username):
    """Initialize OAuth flow for GitHub login/account linking"""
    # Generate state parameter to prevent CSRF
    state = secrets.token_urlsafe(16)

    # Store state and username for verification
    if not hasattr(self, 'oauth_states'):
        self.oauth_states = {}
    self.oauth_states[state] = {
        'username': username,
        'timestamp': time.time()
    }

    # Construct GitHub authorization URL
    params = {
        'client_id': GITHUB_CLIENT_ID,
        'redirect_uri': GITHUB_REDIRECT_URI,
        'scope': 'read:user user:email',
        'state': state
    }
    auth_url = f"{GITHUB_AUTH_URL}?{urllib.parse.urlencode(params)}"
    return auth_url
```

modify the authenticate method to properly handle both authentication methods

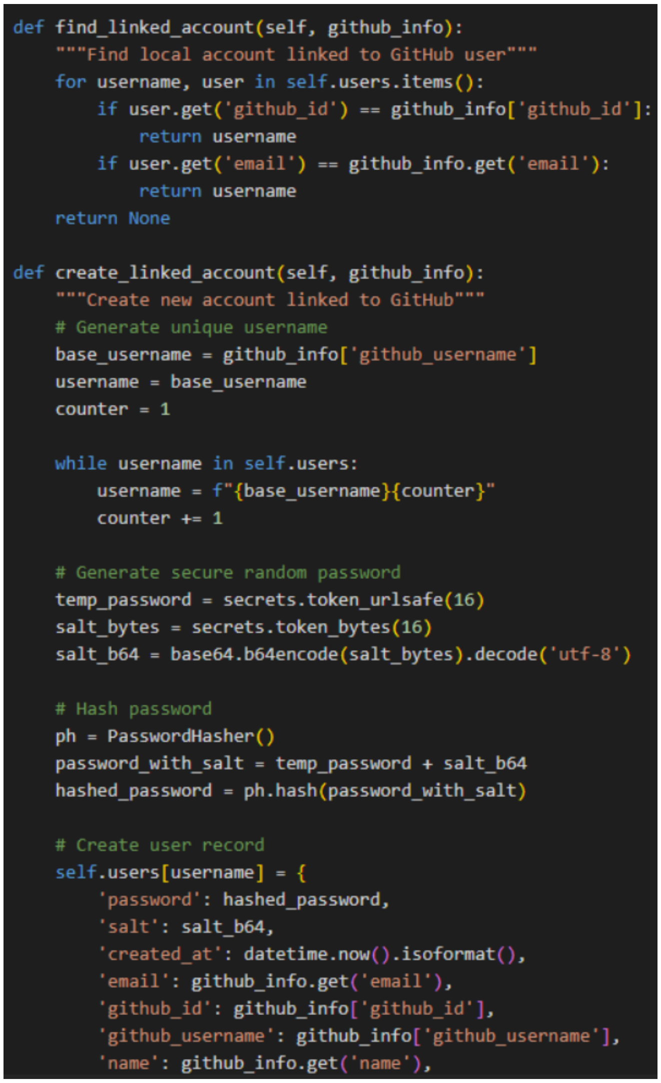


```python
def authenticate(self, username, password=None, totp_code=None, oauth_callback_url=None):
    """Authenticate user with password and/or OAuth"""
    if oauth_callback_url:
        # Handle GitHub OAuth authentication
        success, message, user_info = self.verify_oauth_callback(oauth_callback_url)
        if success:
            # If username provided, try to link accounts
            if username:
                if username in self.users:
                    # Link GitHub account to existing user
                    self.link_github_account(username, user_info)
                    return True, "GitHub account linked successfully"
                else:
                    return False, "Local account not found"

            # Try to find existing linked account
            linked_username = self.find_linked_account(user_info)
            if linked_username:
                return True, f"Logged in via GitHub as {linked_username}"

            # Create new linked account
            new_username, temp_password = self.create_linked_account(user_info)
            return True, f"Created new account: {new_username}"

        return False, message

    # Traditional password authentication
    if not username or not password:
        return False, "Username and password required"

    if username not in self.users:
        return False, "Username not found"

    user = self.users[username]
    ph = PasswordHasher()

    try:
        # Verify password
        salt_b64 = user.get('salt')
```

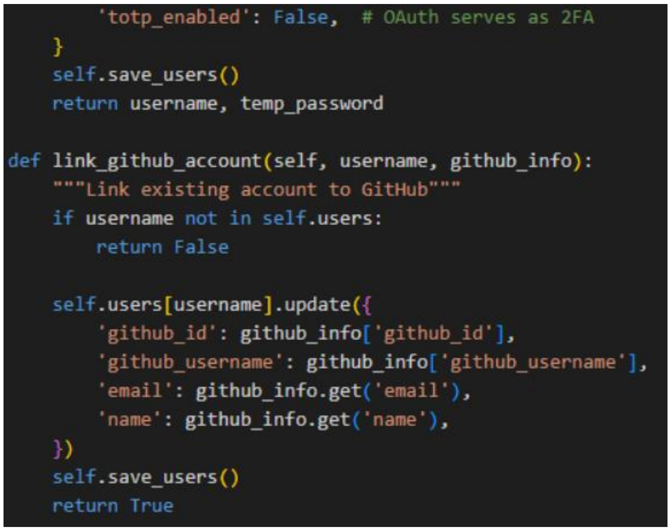


```python
try:
    # Verify password
    salt_b64 = user.get('salt')
    if not salt_b64:
        return False, "Salt missing for user"

    password_with_salt = password + salt_b64
    if not ph.verify(user['password'], password_with_salt):
        return False, "Invalid password"

    # Check if 2FA is required
    if user.get('totp_enabled'):
        if not totp_code:
            return False, "2FA code required"
        if not self.verify_totp(username, totp_code):
            return False, "Invalid 2FA code"

    return True, "Authentication successful"

except Exception as e:
    return False, f"Authentication error: {str(e)}"
```

Add a helper method to link the accounts.

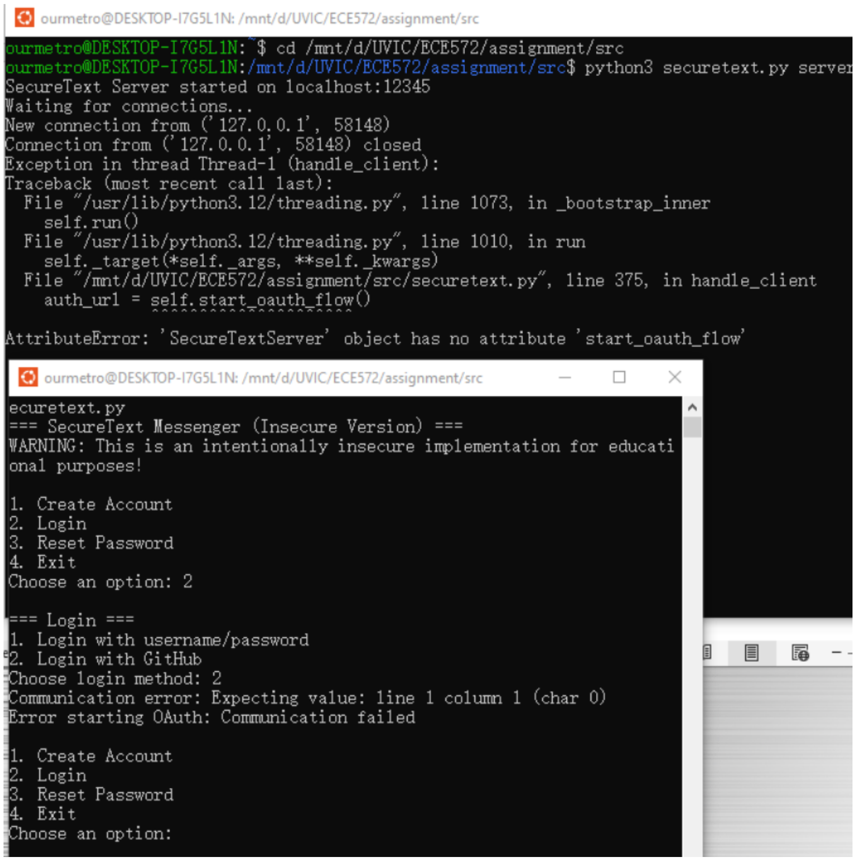


```python
def find_linked_account(self, github_info):
    """Find local account linked to GitHub user"""
    for username, user in self.users.items():
        if user.get('github_id') == github_info['github_id']:
            return username
        if user.get('email') == github_info.get('email'):
            return username
    return None

def create_linked_account(self, github_info):
    """Create new account linked to GitHub"""
    # Generate unique username
    base_username = github_info['github_username']
    username = base_username
    counter = 1

    while username in self.users:
        username = f"{base_username}{counter}"
        counter += 1

    # Generate secure random password
    temp_password = secrets.token_urlsafe(16)
    salt_bytes = secrets.token_bytes(16)
    salt_b64 = base64.b64encode(salt_bytes).decode('utf-8')

    # Hash password
    ph = PasswordHasher()
    password_with_salt = temp_password + salt_b64
    hashed_password = ph.hash(password_with_salt)

    # Create user record
    self.users[username] = {
        'password': hashed_password,
        'salt': salt_b64,
        'created_at': datetime.now().isoformat(),
        'email': github_info.get('email'),
        'github_id': github_info['github_id'],
        'github_username': github_info['github_username'],
        'name': github_info.get('name'),
```



```python
        'totp_enabled': False,  # OAuth serves as 2FA
    }
    self.save_users()
    return username, temp_password

def link_github_account(self, username, github_info):
    """Link existing account to GitHub"""
    if username not in self.users:
        return False

    self.users[username].update({
        'github_id': github_info['github_id'],
        'github_username': github_info['github_username'],
        'email': github_info.get('email'),
        'name': github_info.get('name'),
    })
    self.save_users()
    return True
```

2.2.4 Testing and Validation



```
ourmetro@DESKTOP-I7G5L1N:~$ cd /mnt/d/UVIC/ECE572/assignment/src
ourmetro@DESKTOP-I7G5L1N:/mnt/d/UVIC/ECE572/assignment/src$ python3 securetext.py server
SecureText Server started on localhost:12345
Waiting for connections...
New connection from ('127.0.0.1', 58148)
Connection from ('127.0.0.1', 58148) closed
Exception in thread Thread-1 (handle_client):
Traceback (most recent call last):
  File "/usr/lib/python3.12/threading.py", line 1073, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.12/threading.py", line 1010, in run
    self._target(*self._args, **self._kwargs)
  File "/mnt/d/UVIC/ECE572/assignment/src/securetext.py", line 375, in handle_client
    auth_url = self.start_oauth_flow()
               ^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'SecureTextServer' object has no attribute 'start_oauth_flow'
```

```
ecuretext.py
=== SecureText Messenger (Insecure Version) ===
WARNING: This is an intentionally insecure implementation for educati
onal purposes!

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option: 2

=== Login ===
1. Login with username/password
2. Login with GitHub
Choose login method: 2
Communication error: Expecting value: line 1 column 1 (char 0)
Error starting OAuth: Communication failed

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option:
```

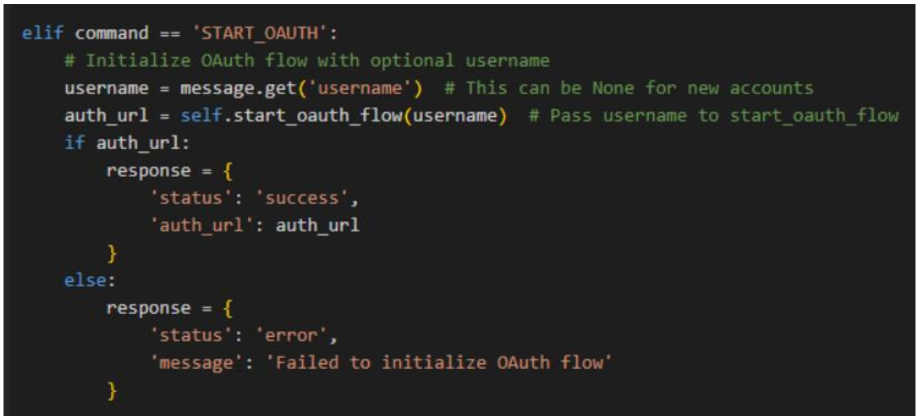
When I test run the program, I encountered this bug, login failed.

The error occurs because we're not properly handling the OAuth flow in the server's handle_client method. The issue is in two places:

The server's handle_client method isn't calling start_oauth_flow correctly

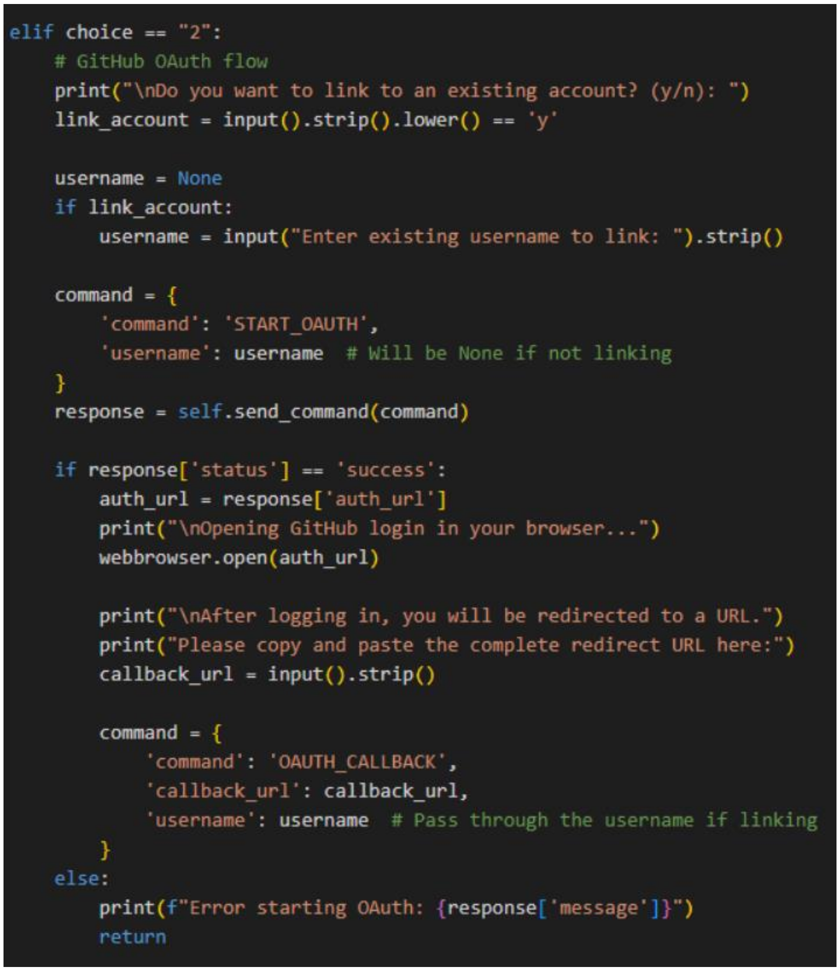
The client's login method needs to handle the username parameter

First, modify the START_OAUTH case in the server's handle_client method:



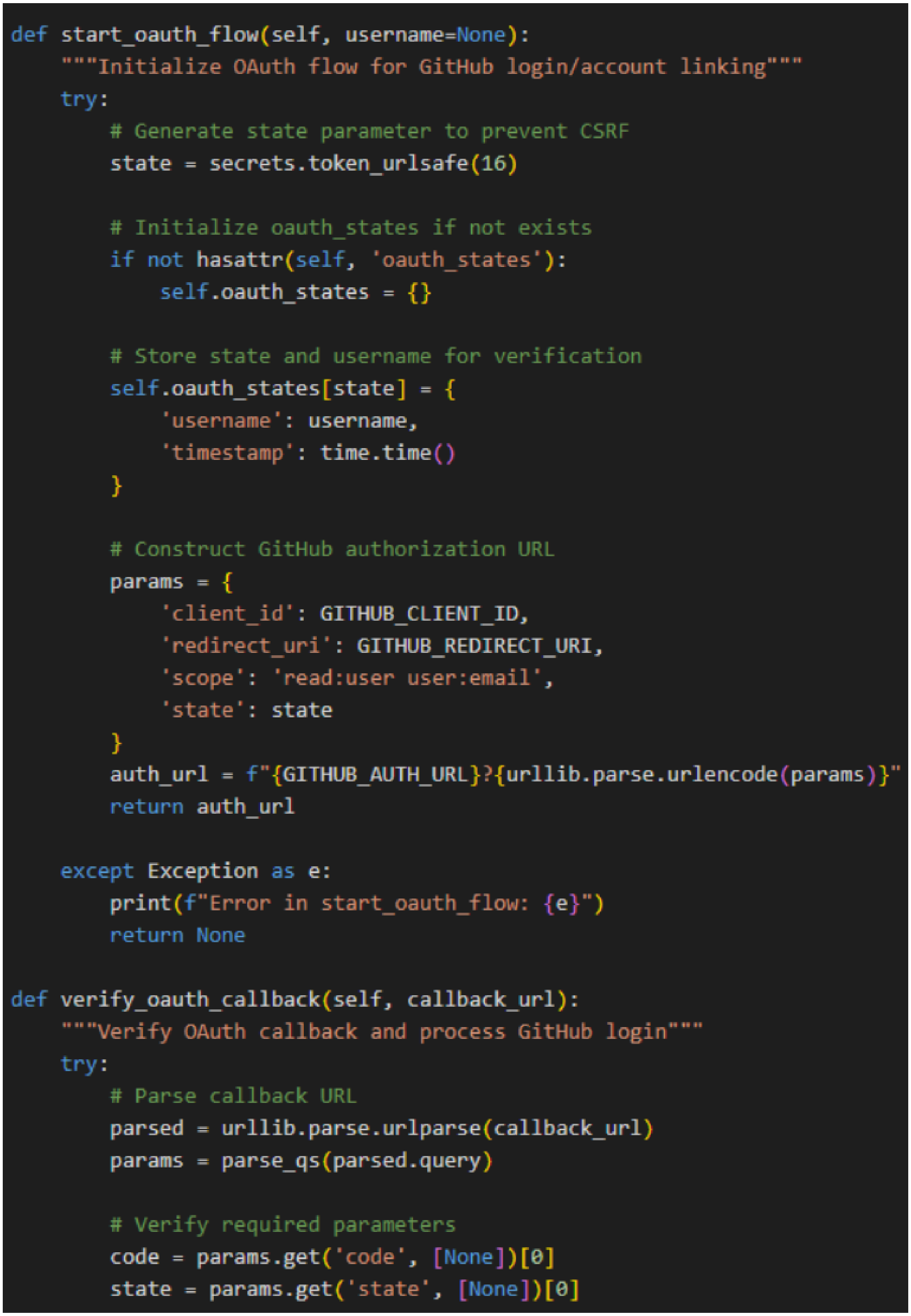
```
elif command == 'START_OAUTH':
    # Initialize OAuth flow with optional username
    username = message.get('username')  # This can be None for new accounts
    auth_url = self.start_oauth_flow(username)  # Pass username to start_oauth_flow
    if auth_url:
        response = {
            'status': 'success',
            'auth_url': auth_url
        }
    else:
        response = {
            'status': 'error',
            'message': 'Failed to initialize OAuth flow'
        }
```

Then update the client's login method to handle the OAuth flow better:

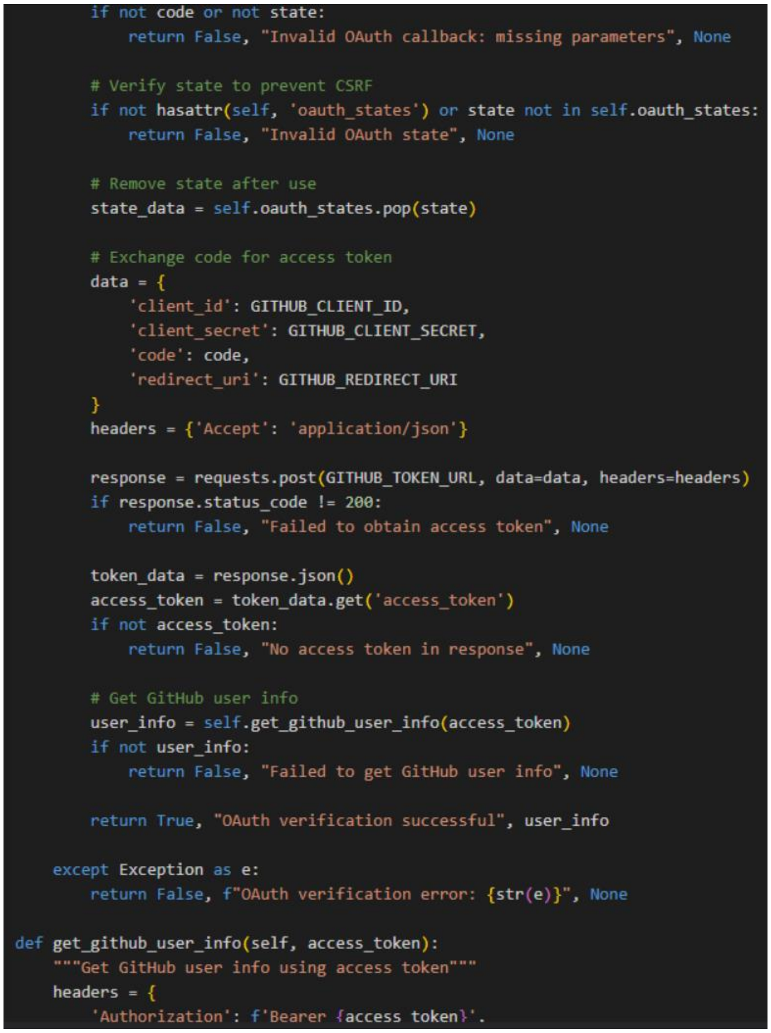


```
elif choice == "2":
    # GitHub OAuth flow
    print("\nDo you want to link to an existing account? (y/n): ")
    link_account = input().strip().lower() == 'y'

    username = None
    if link_account:
        username = input("Enter existing username to link: ").strip()

    command = {
        'command': 'START_OAUTH',
        'username': username  # Will be None if not linking
    }
    response = self.send_command(command)

    if response['status'] == 'success':
        auth_url = response['auth_url']
        print("\nOpening GitHub login in your browser...")
        webbrowser.open(auth_url)

        print("\nAfter logging in, you will be redirected to a URL.")
        print("Please copy and paste the complete redirect URL here:")
        callback_url = input().strip()

        command = {
            'command': 'OAUTH_CALLBACK',
            'callback_url': callback_url,
            'username': username  # Pass through the username if linking
        }
    else:
        print(f"Error starting OAuth: {response['message']}")
        return
```

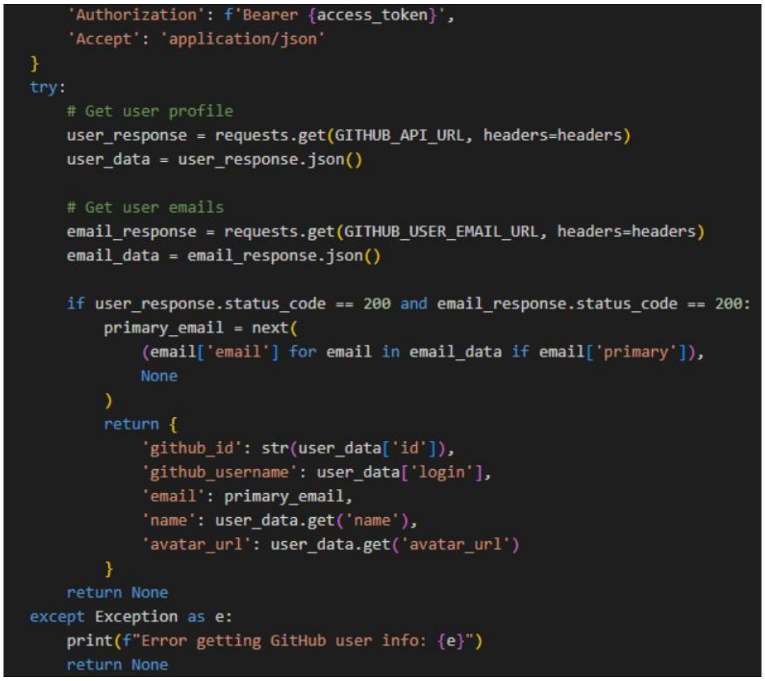
However after the because the start_oauth_flow method is defined in the client class but not in the server class. Move these methods to the SecureTextServer class:



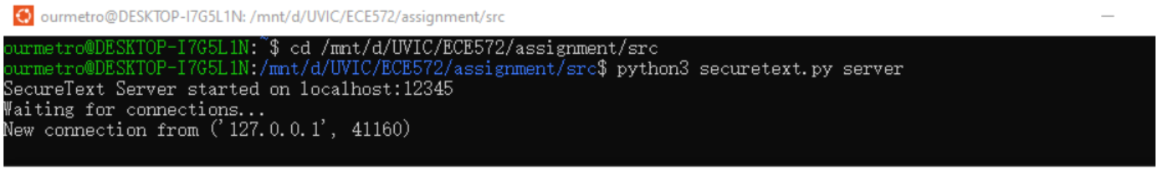
```
def start_oauth_flow(self, username=None):
    """Initialize OAuth flow for GitHub login/account linking"""
    try:
        # Generate state parameter to prevent CSRF
        state = secrets.token_urlsafe(16)

        # Initialize oauth_states if not exists
        if not hasattr(self, 'oauth_states'):
            self.oauth_states = {}

        # Store state and username for verification
        self.oauth_states[state] = {
            'username': username,
            'timestamp': time.time()
        }

        # Construct GitHub authorization URL
        params = {
            'client_id': GITHUB_CLIENT_ID,
            'redirect_uri': GITHUB_REDIRECT_URI,
            'scope': 'read:user user:email',
            'state': state
        }
        auth_url = f"{GITHUB_AUTH_URL}?{urllib.parse.urlencode(params)}"
        return auth_url

    except Exception as e:
        print(f"Error in start_oauth_flow: {e}")
        return None

def verify_oauth_callback(self, callback_url):
    """Verify OAuth callback and process GitHub login"""
    try:
        # Parse callback URL
        parsed = urllib.parse.urlparse(callback_url)
        params = parse_qs(parsed.query)

        # Verify required parameters
        code = params.get('code', [None])[0]
        state = params.get('state', [None])[0]
```



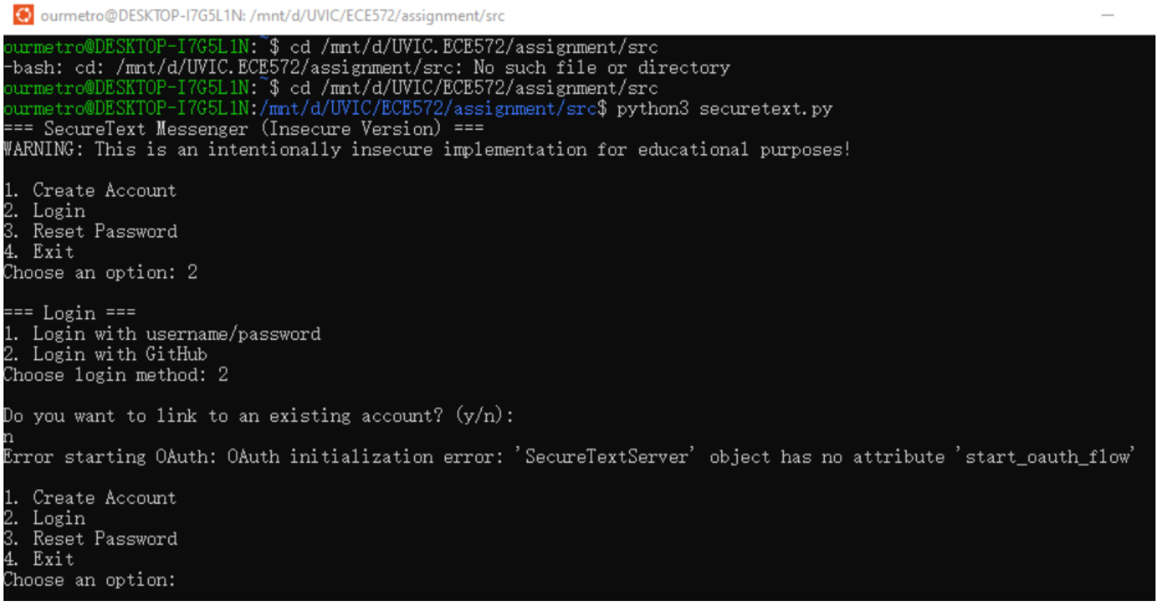
```
        if not code or not state:
            return False, "Invalid OAuth callback: missing parameters", None

        # Verify state to prevent CSRF
        if not hasattr(self, 'oauth_states') or state not in self.oauth_states:
            return False, "Invalid OAuth state", None

        # Remove state after use
        state_data = self.oauth_states.pop(state)

        # Exchange code for access token
        data = {
            'client_id': GITHUB_CLIENT_ID,
            'client_secret': GITHUB_CLIENT_SECRET,
            'code': code,
            'redirect_uri': GITHUB_REDIRECT_URI
        }
        headers = {'Accept': 'application/json'}

        response = requests.post(GITHUB_TOKEN_URL, data=data, headers=headers)
        if response.status_code != 200:
            return False, "Failed to obtain access token", None

        token_data = response.json()
        access_token = token_data.get('access_token')
        if not access_token:
            return False, "No access token in response", None

        # Get GitHub user info
        user_info = self.get_github_user_info(access_token)
        if not user_info:
            return False, "Failed to get GitHub user info", None

        return True, "OAuth verification successful", user_info

    except Exception as e:
        return False, f"OAuth verification error: {str(e)}", None

def get_github_user_info(self, access_token):
    """Get GitHub user info using access token"""
    headers = {
        'Authorization': f'Bearer {access_token}',
```



```
        'Authorization': f'Bearer {access_token}',
        'Accept': 'application/json'
    }
    try:
        # Get user profile
        user_response = requests.get(GITHUB_API_URL, headers=headers)
        user_data = user_response.json()

        # Get user emails
        email_response = requests.get(GITHUB_USER_EMAIL_URL, headers=headers)
        email_data = email_response.json()

        if user_response.status_code == 200 and email_response.status_code == 200:
            primary_email = next(
                (email['email'] for email in email_data if email['primary']),
                None
            )
            return {
                'github_id': str(user_data['id']),
                'github_username': user_data['login'],
                'email': primary_email,
                'name': user_data.get('name'),
                'avatar_url': user_data.get('avatar_url')
            }
        return None
    except Exception as e:
        print(f"Error getting GitHub user info: {e}")
        return None
```



```
ourmetro@DESKTOP-I7G5L1N:~$ cd /mnt/d/UVIC/ECE572/assignment/src
ourmetro@DESKTOP-I7G5L1N:/mnt/d/UVIC/ECE572/assignment/src$ python3 securetext.py server
SecureText Server started on localhost:12345
Waiting for connections...
New connection from ('127.0.0.1', 41160)
```



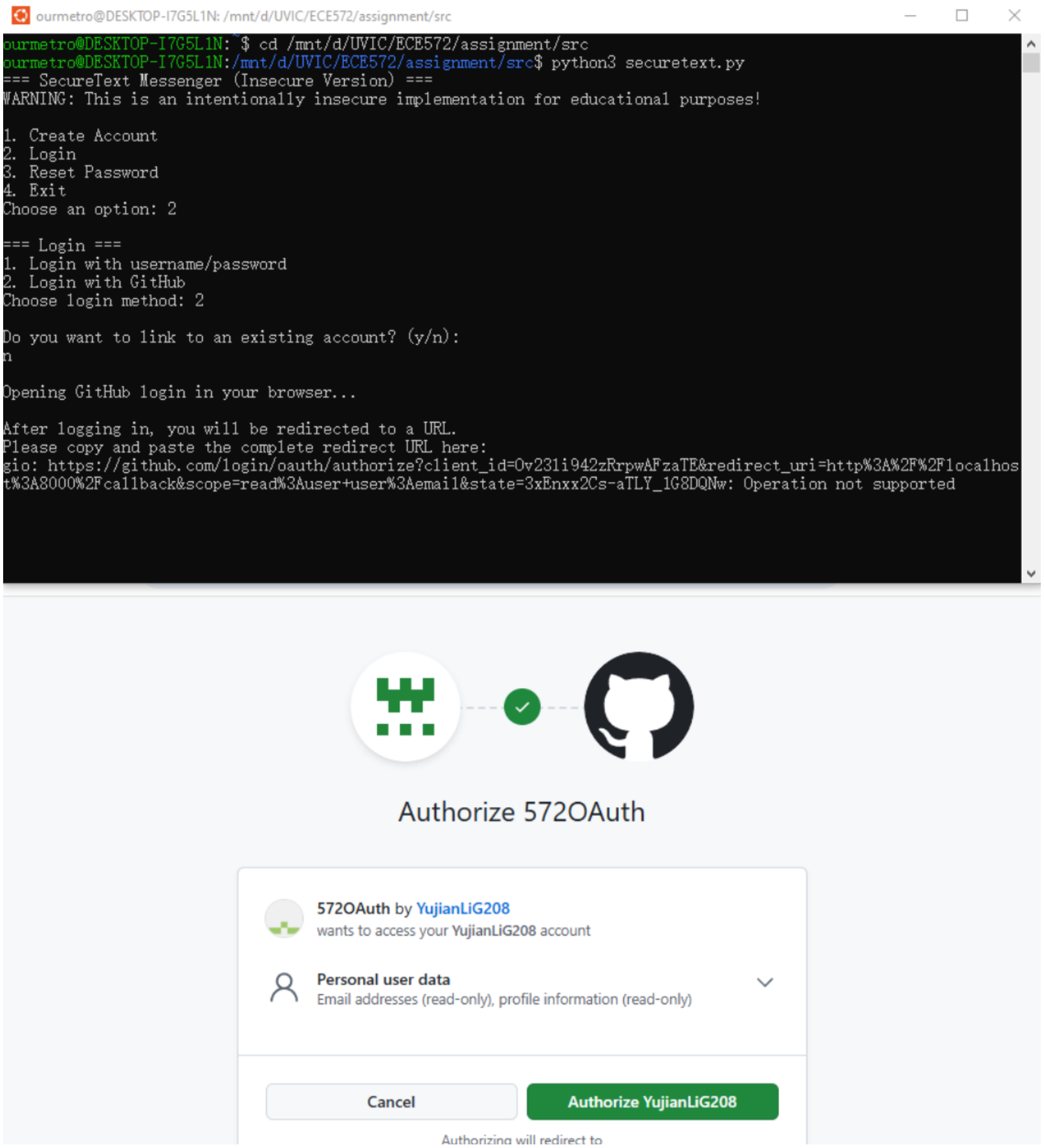
```
ourmetro@DESKTOP-I7G5L1N:~$ cd /mnt/d/UVIC.ECE572/assignment/src
-bash: cd: /mnt/d/UVIC.ECE572/assignment/src: No such file or directory
ourmetro@DESKTOP-I7G5L1N:~$ cd /mnt/d/UVIC/ECE572/assignment/src
ourmetro@DESKTOP-I7G5L1N:/mnt/d/UVIC/ECE572/assignment/src$ python3 securetext.py
=== SecureText Messenger (Insecure Version) ===
WARNING: This is an intentionally insecure implementation for educational purposes!

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option: 2

=== Login ===
1. Login with username/password
2. Login with GitHub
Choose login method: 2

Do you want to link to an existing account? (y/n):
n
Error starting OAuth: OAuth initialization error: 'SecureTextServer' object has no attribute 'start_oauth_flow'

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option:
```

However, the problem still exist.

I moved all the OAuth helper methods from SecureTextClient class to the SecureTextServer class, and the problem finally solved. I am able to sign in to the Securetext with my github account.

However, I still need to copy-paste the link from my WSL to my WIN. And because redirect URI is still a placeholder http://localhost as instructed in the assignment guidance.

This log in function is still a “demo only” version.



```
ourmetro@DESKTOP-I7G5L1N:~$ cd /mnt/d/UVIC/ECE572/assignment/src
ourmetro@DESKTOP-I7G5L1N:/mnt/d/UVIC/ECE572/assignment/src$ python3 securetext.py
=== SecureText Messenger (Insecure Version) ===
WARNING: This is an intentionally insecure implementation for educational purposes!

1. Create Account
2. Login
3. Reset Password
4. Exit
Choose an option: 2

=== Login ===
1. Login with username/password
2. Login with GitHub
Choose login method: 2

Do you want to link to an existing account? (y/n):
n

Opening GitHub login in your browser...

After logging in, you will be redirected to a URL.
Please copy and paste the complete redirect URL here:
gio: https://github.com/login/oauth/authorize?client_id=Ov231i942zRrpwAFzaTE&redirect_uri=http%3A%2F%2Flocalhos
t%3A8000%2Fcallback&scope=read%3Auser+user%3Aemail&state=3xEnxx2Cs-aTLY_1G8DQNw: Operation not supported
```

Authorize 572OAuth

572OAuth by YujianLiG208
wants to access your YujianLiG208 account

Personal user data
Email addresses (read-only), profile information (read-only)

——————————————————————————————

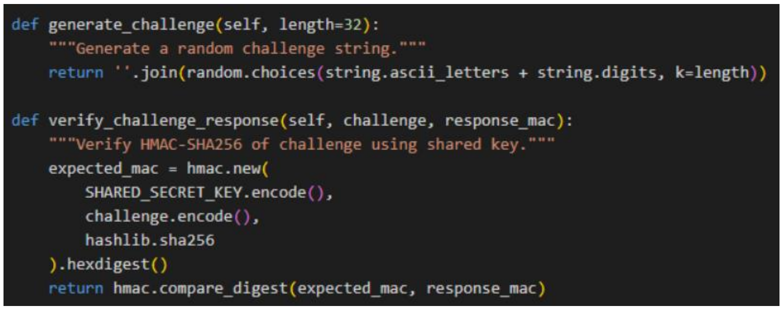
2.3 Task 6: Zero Trust Implementation

2.3.1 Objective

Implement challenge-response protocols and Zero Trust principles.

2.3.2 Implementation Details

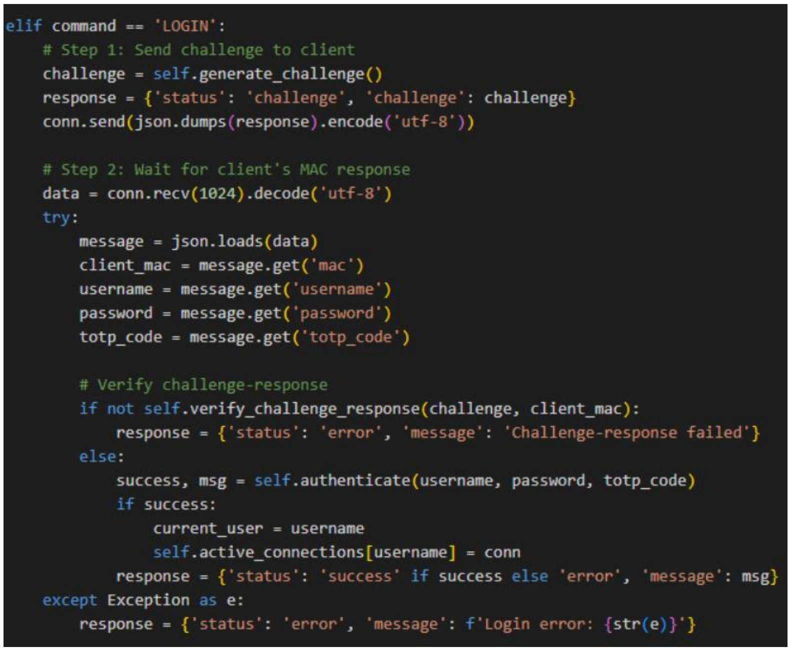
Add challenge generation and verification methods to the SecureTextServer



```python
def generate_challenge(self, length=32):
    """Generate a random challenge string."""
    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))

def verify_challenge_response(self, challenge, response_mac):
    """Verify HMAC-SHA256 of challenge using shared key."""
    expected_mac = hmac.new(
        SHARED_SECRET_KEY.encode(),
        challenge.encode(),
        hashlib.sha256
    ).hexdigest()
    return hmac.compare_digest(expected_mac, response_mac)
```

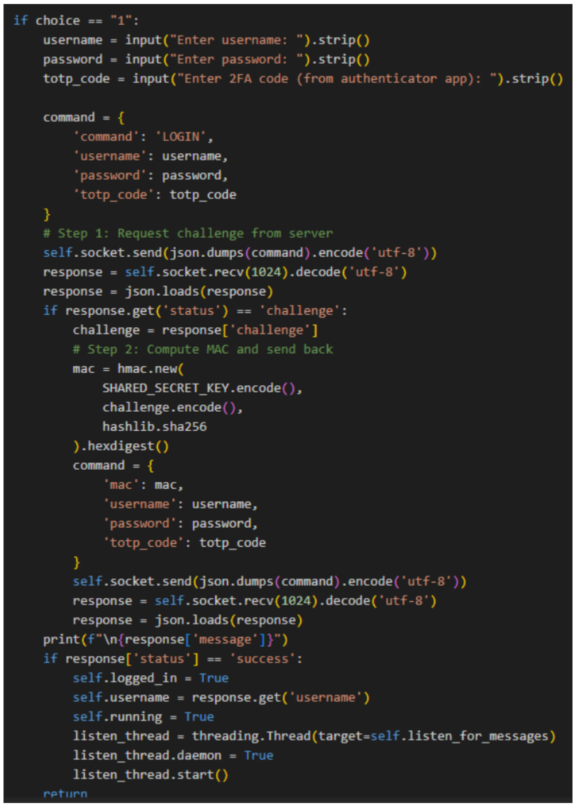
Log in handling enhanced with 3 steps:

Send challenge to client. Wait for client's MAC response, and verify challenge-response



```python
elif command == 'LOGIN':
    # Step 1: Send challenge to client
    challenge = self.generate_challenge()
    response = {'status': 'challenge', 'challenge': challenge}
    conn.send(json.dumps(response).encode('utf-8'))

    # Step 2: Wait for client's MAC response
    data = conn.recv(1024).decode('utf-8')
    try:
        message = json.loads(data)
        client_mac = message.get('mac')
        username = message.get('username')
        password = message.get('password')
        totp_code = message.get('totp_code')

        # Verify challenge-response
        if not self.verify_challenge_response(challenge, client_mac):
            response = {'status': 'error', 'message': 'Challenge-response failed'}
        else:
            success, msg = self.authenticate(username, password, totp_code)
            if success:
                current_user = username
                self.active_connections[username] = conn
            response = {'status': 'success' if success else 'error', 'message': msg}
    except Exception as e:
        response = {'status': 'error', 'message': f'Login error: {str(e)}'}
```

On the client side(SecureTextClient), set up the similar functions



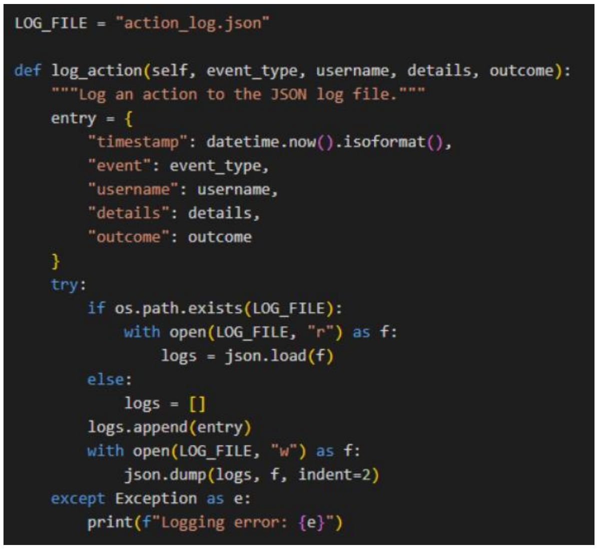
```python
if choice == "1":
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()
    totp_code = input("Enter 2FA code (from authenticator app): ").strip()

    command = {
        'command': 'LOGIN',
        'username': username,
        'password': password,
        'totp_code': totp_code
    }
    # Step 1: Request challenge from server
    self.socket.send(json.dumps(command).encode('utf-8'))
    response = self.socket.recv(1024).decode('utf-8')
    response = json.loads(response)
    if response.get('status') == 'challenge':
        challenge = response['challenge']
        # Step 2: Compute MAC and send back
        mac = hmac.new(
            SHARED_SECRET_KEY.encode(),
            challenge.encode(),
            hashlib.sha256
        ).hexdigest()
        command = {
            'mac': mac,
            'username': username,
            'password': password,
            'totp_code': totp_code
        }
        self.socket.send(json.dumps(command).encode('utf-8'))
        response = self.socket.recv(1024).decode('utf-8')
        response = json.loads(response)
    print(f"\n{response['message']}")
    if response['status'] == 'success':
        self.logged_in = True
        self.username = response.get('username')
        self.running = True
        listen_thread = threading.Thread(target=self.listen_for_messages)
        listen_thread.daemon = True
        listen_thread.start()
    return
```

Next implementation is action logging for authentication, commands, and role-based access, with warnings for repeated failed logins

All authentication attempts, commands, and role-based access denials are logged to action_log.json.

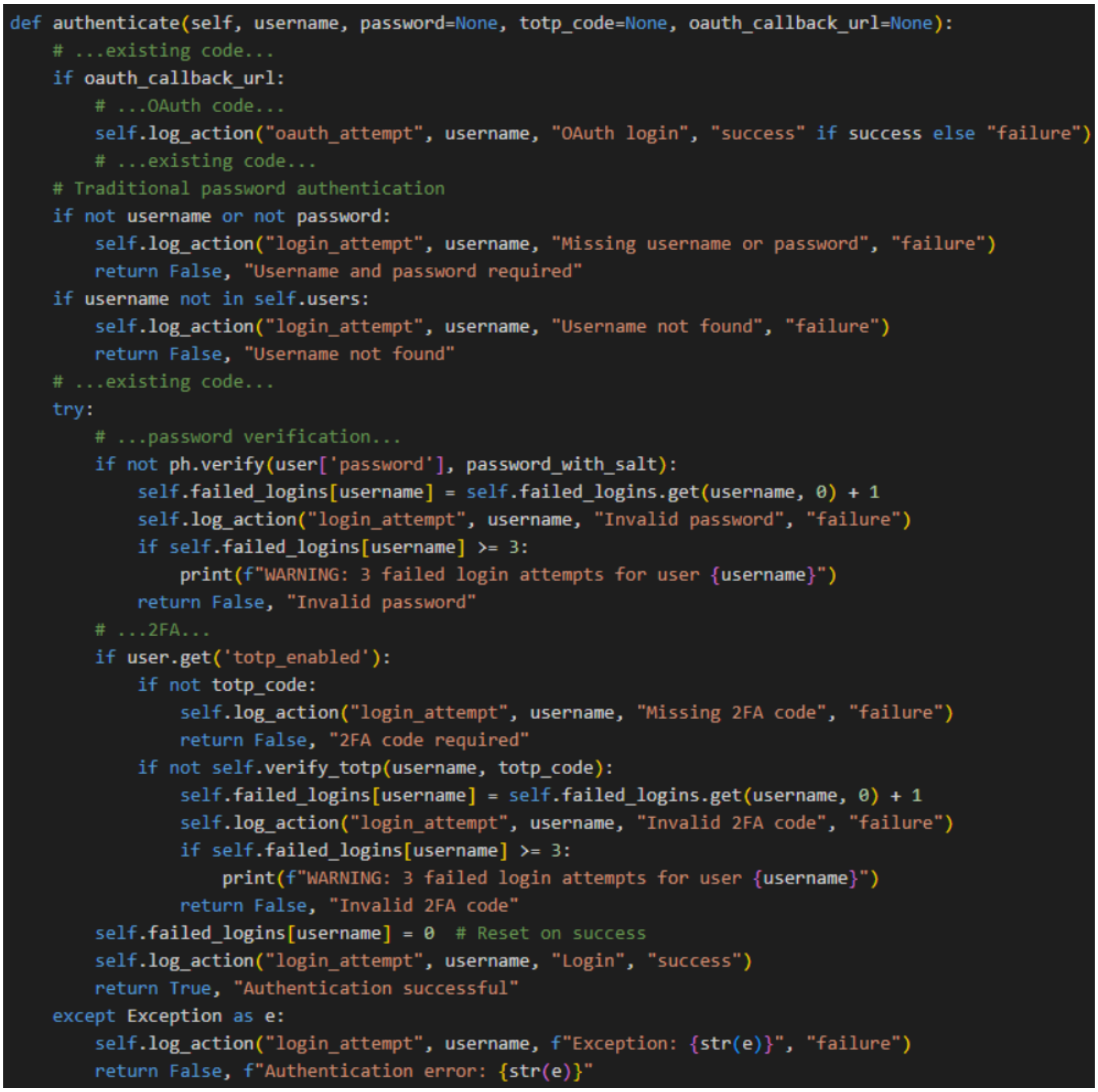
After 3 failed logins, a warning is printed.

The log_action, serve as logging utility is being added to SecureTextServer class.



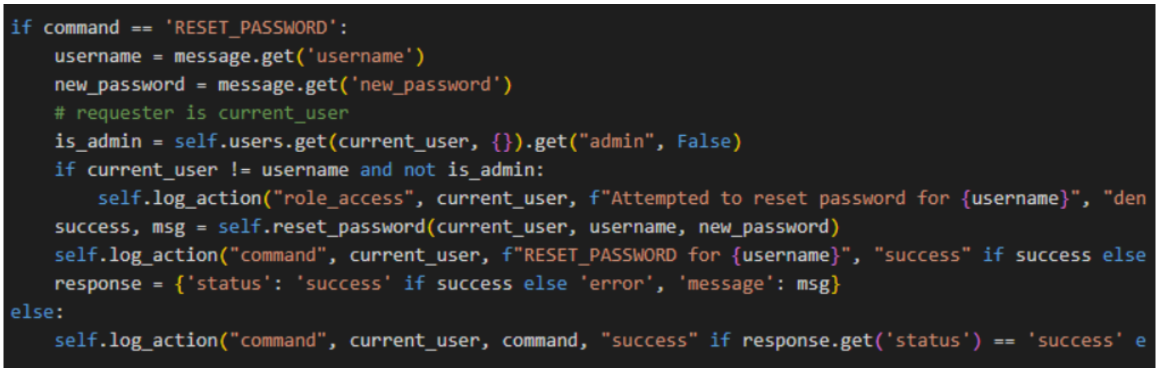
```python
LOG_FILE = "action_log.json"

def log_action(self, event_type, username, details, outcome):
    """Log an action to the JSON log file."""
    entry = {
        "timestamp": datetime.now().isoformat(),
        "event": event_type,
        "username": username,
        "details": details,
        "outcome": outcome
    }
    try:
        if os.path.exists(LOG_FILE):
            with open(LOG_FILE, "r") as f:
                logs = json.load(f)
        else:
            logs = []
        logs.append(entry)
        with open(LOG_FILE, "w") as f:
            json.dump(logs, f, indent=2)
    except Exception as e:
        print(f"Logging error: {e}")
```

Log authentication attempts and warn after 3 failures



```
def authenticate(self, username, password=None, totp_code=None, oauth_callback_url=None):
    # ...existing code...
    if oauth_callback_url:
        # ...OAuth code...
        self.log_action("oauth_attempt", username, "OAuth login", "success" if success else "failure")
        # ...existing code...
    # Traditional password authentication
    if not username or not password:
        self.log_action("login_attempt", username, "Missing username or password", "failure")
        return False, "Username and password required"
    if username not in self.users:
        self.log_action("login_attempt", username, "Username not found", "failure")
        return False, "Username not found"
    # ...existing code...
    try:
        # ...password verification...
        if not ph.verify(user['password'], password_with_salt):
            self.failed_logins[username] = self.failed_logins.get(username, 0) + 1
            self.log_action("login_attempt", username, "Invalid password", "failure")
            if self.failed_logins[username] >= 3:
                print(f"WARNING: 3 failed login attempts for user {username}")
            return False, "Invalid password"
        # ...2FA...
        if user.get('totp_enabled'):
            if not totp_code:
                self.log_action("login_attempt", username, "Missing 2FA code", "failure")
                return False, "2FA code required"
            if not self.verify_totp(username, totp_code):
                self.failed_logins[username] = self.failed_logins.get(username, 0) + 1
                self.log_action("login_attempt", username, "Invalid 2FA code", "failure")
                if self.failed_logins[username] >= 3:
                    print(f"WARNING: 3 failed login attempts for user {username}")
                return False, "Invalid 2FA code"
        self.failed_logins[username] = 0  # Reset on success
        self.log_action("login_attempt", username, "Login", "success")
        return True, "Authentication successful"
    except Exception as e:
        self.log_action("login_attempt", username, f"Exception: {str(e)}", "failure")
        return False, f"Authentication error: {str(e)}"
```

Log user commands and role-based access



```
if command == 'RESET_PASSWORD':
    username = message.get('username')
    new_password = message.get('new_password')
    # requester is current_user
    is_admin = self.users.get(current_user, {}).get("admin", False)
    if current_user != username and not is_admin:
        self.log_action("role_access", current_user, f"Attempted to reset password for {username}", "den
    success, msg = self.reset_password(current_user, username, new_password)
    self.log_action("command", current_user, f"RESET_PASSWORD for {username}", "success" if success else
    response = {'status': 'success' if success else 'error', 'message': msg}
else:
    self.log_action("command", current_user, command, "success" if response.get('status') == 'success' e
```

2.3.3 Challenges and Solutions

<!-- What problems did you encounter and how did you solve them? -->

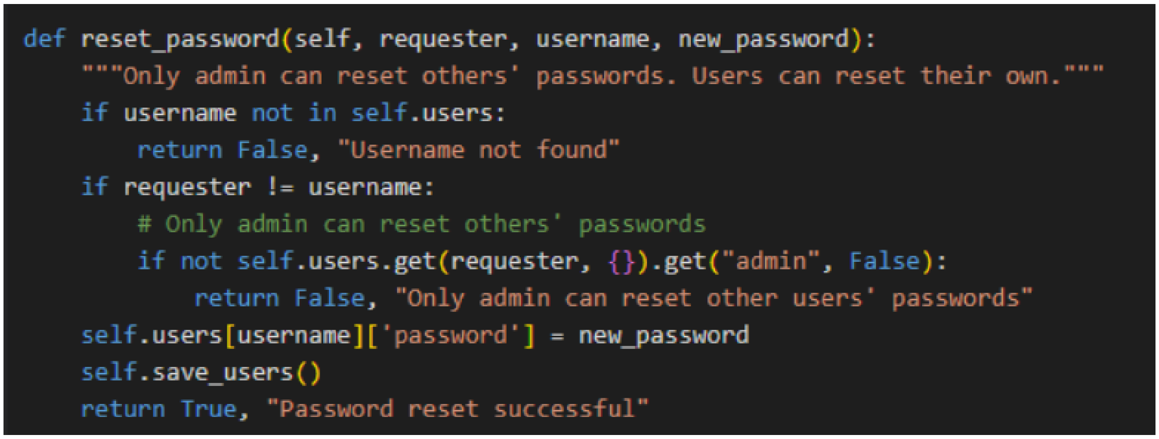
I faced a lot of problem when setting up the role-based control, as it includes several different steps. At the beginning I had completely no idea what to do, but considering the phased working method, I roughly had a framework.

I decides to use Boolean value for admins.

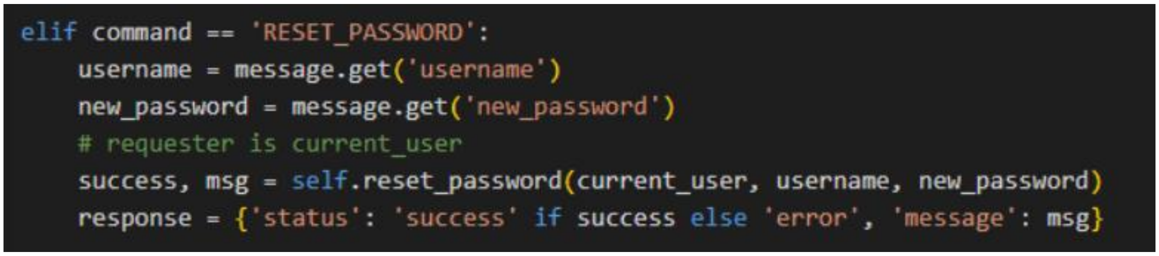
admin: true for administrators, admin: false for regular users. Then I slightly modified create_account as def create_account(self, username, password, admin=False)

New accounts are not administrator as default.

The next step is to restrict password reset of other users to only admins.



```python
def reset_password(self, requester, username, new_password):
    """Only admin can reset others' passwords. Users can reset their own."""
    if username not in self.users:
        return False, "Username not found"
    if requester != username:
        # Only admin can reset others' passwords
        if not self.users.get(requester, {}).get("admin", False):
            return False, "Only admin can reset other users' passwords"
    self.users[username]['password'] = new_password
    self.save_users()
    return True, "Password reset successful"
```



```python
elif command == 'RESET_PASSWORD':
    username = message.get('username')
    new_password = message.get('new_password')
    # requester is current_user
    success, msg = self.reset_password(current_user, username, new_password)
    response = {'status': 'success' if success else 'error', 'message': msg}
```

2.3.4 Testing and Validation

I found a huge problem about my approach to add role based access control.

Setting up any new admin account would requires me to change the original code and then reset it back. If I want to add admin privilege to existing users, I would need to manually open the users.json file and edit in line.

This overly clumsy approach is obviously not feasible and prevents the implementation of verification, but it is also difficult for me to implement better ideas.

———————————————————————————

3. Security Analysis

3.1 Vulnerability Assessment

Analyze security improvements made

3.2 Security Improvements

Before vs. After Analysis:

- **Authentication**: Before: once the attacker have the password and user name they will be able to login to the account. After: 2nd authentication required which means attackers cannot login to the account even if they have the password and username.
- **Data Protection**: After applying the challenge-response authentication, the server will only allow login if the client proves knowledge of the shared secret by correctly responding to the challenge.

Let's make a compare of TOTP and challenge-response, summarized with ChatGPT

- **TOTP**
  How it works: Both sides share a fixed secret key. Each 30 s (typical) they hash *key time-counter* with HMAC-SHA-1 and truncate to a 6-digit code. The user reads the code from an app or token and types it during login.
  Strengths: Works offline, cheap to deploy, independent of the network or server availability once the seed is provisioned. Codes expire quickly, limiting replay.
  Weaknesses: Relies on clock sync and a tolerance window; phishing sites can relay the typed code in real time; brute-force space grows when servers widen the window; seed theft compromises all future codes.
- **Challenge-Response**
  How it works: During each login the server sends a fresh, random challenge. The authenticator computes a cryptographic response (e.g., HMAC with a symmetric key, digital signature with a private key, or password-derived hash) and returns it. The server verifies locally.
  Strengths: No need for time sync; each response is bound to a unique nonce, rendering replay and real-time phishing ineffective. With public-key variants (FIDO, smartcards) the secret never leaves the device, so credential theft is far harder.
  Weaknesses: Requires an on-line exchange and usually specialized hardware or software; more complex to implement and provision than TOTP.

- Basic ID+ password login will make attackers easily gain access to accounts and sensitive information. TOTP is simple and ubiquitous for 2FA, but its manual code entry may be phishable. Challenge-response offers stronger binding and anti-phishing properties, making it the preferred choice for high-risk or enterprise environments despite higher deployment overhead.

3.3 Threat Model

Use the following security properties and threat actors in your threat modeling. You can add extra if needed.

Threat Actors:

9. Passive Network Attacker: Can intercept but not modify traffic
10. Active Network Attacker: Can intercept and modify traffic
11. Malicious Server Operator: Has access to server and database
12. Compromised Client: Attacker has access to user's device (However, they must have *physical* control to the registered 2nd FA application phone and being able to use it)

Security Properties Achieved:

- [ ] Confidentiality
- [ ] Integrity
- [ ] Authentication
- [ ] Authorization
- [ ] Non-repudiation
- [ ] Perfect Forward Secrecy
- [ ] Privacy

————————————————————————

4. Attack Demonstrations

4.1 Attack 1: SIM swapping attack

4.1.1 Objective

Authentication, social engineering.

4.1.2 Attack Setup

NOTE: I do not have a spare SIM card or personal account, it is also impossible to get someone else’s SIM card to practice since impersonating someone else **is a crime**.

Attackers start by gathering as much personal information about the target person as they can from social media, dark web, leaked database, and directly through phishing.

4.1.3 Attack Execution

After attackers successfully fake the identity of the victim, they use the information to pretend they are the victim, to gain access to the victim's account and transfer victim's mobile number to a different SIM card or eSIM under their control. Once they've "swapped" the SIM, calls and SMS will go to the victim's phone number route to the phone in the criminal's control.

If a thief in a criminal gang steals a mobile phone, in the time window before the owner discovers the theft and reports the loss to the wireless mobile service operator, other members in the gang can remove the physical SIM card from the stolen phone and insert it into their phones to achieve the same effect.

4.1.4 Results and Evidence

At that point, the criminals can use their phone to receive one-time security codes or calls that banks and other companies use to safeguard customer accounts. The victims might not know this has even happened until the phone number no longer works or other accounts being banned for scam activities. All accounts bound to the mobile number will be stolen, and the attacker can reset the password or even directly access account by receiving SMS.

4.1.5 Mitigation

In addition to raising awareness of phishing scams, implementing TOTP authentication apps as 2nd FA will directly eliminate the risk of SIM swapping attack. TOTP apps replace the insecure, carrier-mediated channel with cryptographically generated codes under the user's direct custody. Even if the password is compromised, the attacker cannot access the account due to the mandatory second step verification after entering the password. For business IT operation, apply smartcard and strict card management policy (not allowed to bring outside the office) is the best 2nd FA solution against any form of phishing and social engineering.

TOTP extends HOTP by replacing its event counter with a time counter. Both client and server share a 160-bit secret key K. Every X seconds (RFC 6238's default is 30 s) they compute $T=\lfloor(t_{unix}-t_0)/X\rfloor$ and feed $K\|T$ into HMAC-SHA-1. The keyed-hash output (20 bytes) is dynamically truncated to 31 bits and reduced mod 10^d (usually 10^6), producing a short decimal code. HMAC provides pre-image resistance—seeing many codes does not reveal KKK—while the changing TTT prevents replay outside the current window. Although SHA-1 is collision-weakened, its role inside HMAC with a random key remains safe for six-digit OTPs, and stronger hashes (SHA-256/512) are a drop-in upgrade.

Because TOTP's correctness hinges on clock alignment, RFC 6238 recommends accepting codes for the current step and one step either side (±30 s) to absorb small skew . Some deployments widen the *tolerance window* to ±2–3 steps when user devices have poor timekeeping, but every extra step triples an attacker's online guessing space.

4.2 Attack 2: OAuth Vulnerabilities

4.2.1 Objective

Authentication. Let's discuss two attack scenario here: Authorization-Code Interception (A), Redirect-URI Manipulation (R). Key points are listed by ChatGPT and then I organized the research

4.2.2 Attack Setup

(A) Attackers hijacked an open WIFI hotspot, silently forces all HTTP traffic through the attacker's proxy. The victim connect his/her phone to that hotspot, launches a single-page web app that uses the classic OAuth 2.0 authorization-code flow without PKCE; the app receives the code in a front-channel redirect (https://app.example/callback?code=ABC123).

(R) A company registers its OAuth client with the authorization server using a permissive wildcard: https://*.startapp.io/auth/callback. The attacker buys evil.startapp.io, hosts a look-alike landing page, and adds the same path.

4.2.3 Attack Execution

(A) When the authorization server redirects the browser, the proxy reads the visible code parameter and copies it before forwarding the packet. Within seconds, the attacker's script posts code=ABC123—along with the client's public credentials—to the token endpoint.

(R) The attacker crafts an authorization URL that lists their rogue domain as the redirect_uri and sends a phishing email. The unsuspecting user signs in at the genuine IdP and grants permission. The IdP dutifully redirects the authorization code to https://evil.startapp.io/auth/callback, delivering it straight to the attacker.

4.2.4 Results and Evidence

(A) The server issues a valid access token tied to the victim's scopes. The attacker now calls protected APIs, views private data, and even refreshes the session, while the victim continues unaware on the compromised network.

(R) The attacker exchanges the code for tokens, gaining API access equal to the victim's consents. Meanwhile, the legitimate app never sees the callback and the user assumes the login simply failed.

4.2.5 Mitigation

(A) Enable Proof Key for Code Exchange (PKCE) so every authorization request includes a code_challenge. The intercepted code alone becomes useless without the one-time code_verifier kept inside the JavaScript runtime. Enforce HTTPS, move the code exchange to a confidential back-end, and monitor token-endpoint rate limits for anomalies.

(R) Pre-register a fully qualified URL, use exact redirect-URI matching with no wildcards or variable paths. Validate the redirect_uri parameter against this whitelist at both the client and IdP, and require re-registration plus proof-of-domain control for any new callback addresses.

—————————————————————————————

5. Performance Evaluation

Basic test results in terms of resources used in terms of hardware and time. Also, if the test has limitations and fix worked properly(test passed or failed)

Not applicable. No noticeable performance issues in development and test environments (my laptop)

—————————————————————————————

6. Lessons Learned

6.1 Technical Insights

How to setup TOTP

Introduce 2ND FA OAuth with github

Practice 0 trust.

Define roles: user, admin. Grant privilege to admin.

6.2 Security Principles

Applied Principles:

- **Defense in Depth**: [How you applied this]
- **Least Privilege**: [How you applied this]
- **Fail Secure**: [How you applied this]
- **Economy of Mechanism**: [How you applied this]

—————————————————————————————

7. Conclusion

7.1 Summary of Achievements

Set up TOTP authentication

Introduce OAuth as 2nd FA

Apply 0 trust, set up role-based access control. Logging setup

7.2 Security and Privacy Posture Assessment

Remaining Vulnerabilities:

- Vulnerability 1: Admin

Suggest an Attack: If the admin account is compromised, the hacker will have access to all user's information, which means have full control over securetext.

7.3 Future Improvements

What would I do if I had more time?

13. Improvement 1: Actually make OAuth github sign in functional instead of stick to the instruction.
14. Improvement 2: If can get external teammates (such as collaborating with other students), I will make the log only available for admin, and improve the role setup for a more practical solution.

8. Reference

[1] "What is a SIM Swapping Scam? Protect Your Device Against SIM Hackers," www.verizon.com. https://www.verizon.com/about/account-security/sim-swapping

[2] Daryush Purmostafa, "What is TOTP and RFC 6238? | Medium," Medium, Oct. 11, 2023. https://medium.com/@daryush.purmostafa/what-is-totp-a3dcd0ad315e

[3] "Time-based One-Time Password," Wikipedia, Mar. 10, 2021. https://en.wikipedia.org/wiki/Time-based_One-Time_Password

[4] "WorkOS," Workos.com, 2025. https://workos.com/blog/oauth-common-attacks-and-how-to-prevent-them