Secure Authentication - Information Security CS3002

Hashim Muhammad Nadeem (21i-1675)
Muhammad Huzaifa Khan (21i-2689)
Wajeeh Ul Hasan Rajput (21i-2684)
Najam Ali Abbas (20i-2612)

October 15, 2024

1 Iteration 1 Recap: Foundational Setup

In **Iteration 1**, we established the core foundation of the secure authentication system. Here's a summary of the achievements:

1. **Backend Setup with FastAPI**:
   - Implemented endpoints for user registration (/signup), login (/login), and OTP verification (/verify-otp).
   - Connected FastAPI with PostgreSQL (via Supabase) for user data storage and OTP management.
2. **Password Encryption**:
   - Used bcrypt with salting to securely store passwords in the database.
   - Added functionality to verify hashed passwords during login attempts.
3. **OTP System**:
   - Developed an OTP generation system using the `secrets` module.
   - Set up email functionality using SMTP (Gmail) to send the OTP to users' registered email addresses.
   - Implemented OTP expiration logic (5-minute timeout).
4. **Frontend Code (Not Integrated Yet)**:
   - Developed basic frontend forms for user registration and login (HTML/CSS), awaiting backend integration.

2 Iteration 2: Session Management and Frontend Integration

Objective: In **Iteration 2**, we focus on managing user sessions for a seamless login experience and integrating the frontend with the backend. Additionally, we will enhance the frontend's aesthetics using frameworks like Bootstrap.

2.1 Tasks

1. **Session Management**:
   - **JWT Authentication**: Implement session management using JSON Web Tokens (JWT). After a successful OTP verification, generate a JWT token for the user to maintain session states.
     - **Endpoint**: Update /login to issue a JWT upon successful login and OTP verification.
     - **Token Expiration**: Set a token expiration time (e.g., 30 minutes or 1 hour).
     - **Token Revocation**: Add functionality for token revocation or session termination (logout).
   - **Authorization Middleware**: Protect secure routes by requiring a valid JWT to access certain endpoints, ensuring that only logged-in users can perform actions like updating their profile.
2. **Frontend Integration**:
   - Connect the developed registration and login forms to the FastAPI backend using Fetch API or Axios in JavaScript.
   - Capture user input (username, email, password) and send it to the backend via API calls.
   - Capture the OTP sent to the user's email and implement the frontend logic to allow the user to input and verify the OTP.
3. **Frontend Beautification**:
   - Use Bootstrap or a similar CSS framework to enhance the aesthetics of the forms (signup, login, and OTP input pages).
   - Add input validation on the client side for better user experience (e.g., checking password strength before submission).
   - Improve UX/UI for error handling (e.g., clear messages for failed OTP verification or incorrect passwords).
4. **Logout Functionality**:
   - Create a **logout** route that invalidates the user's session by deleting the JWT token from the client side (e.g., remove it from local storage or cookies).

2.2 Deliverables for Iteration 2

1. **Session Management with JWT**: JWT authentication system integrated for managing user sessions. Secure route protection using JWT-based authorization.
2. **Frontend-Backend Integration**: Fully functional frontend connected with the backend for signup, login, and OTP verification.
3. **Frontend Enhancement**: Enhanced user experience through Bootstrap and form validation.

2.3 Challenges to Anticipate

- Handling token expiration and ensuring a smooth user experience when tokens expire (e.g., redirecting users to login).
- Ensuring secure storage of JWT tokens (e.g., using HTTP-only cookies to avoid XSS attacks).

3 Iteration 3: Google OAuth2 Integration and Advanced Security Features

Objective: In **Iteration 3**, we enhance the authentication system by adding **Google OAuth2** as an alternative login method and implement additional security measures like rate-limiting and CSRF protection.

3.1 Tasks

1. **Google OAuth2 Integration**:
   - Set up Google OAuth2 API credentials in Google Cloud Console and use FastAPI's **OAuth2 middleware** to handle Google authentication.
   - Add an option on the login page to "Sign in with Google."
   - Retrieve the user's email and profile information (e.g., name) after successful authentication with Google and store it in the database if the user logs in for the first time.
   - Generate a JWT token for session management.
2. **Rate Limiting**:
   - Implement rate limiting to prevent brute-force attacks on login and OTP endpoints (e.g., limit OTP requests to 3 per hour per user).
   - Use **FastAPI RateLimiter** or Redis to manage rate-limiting logic.
3. **Security Enhancements**:

- Implement **CSRF Protection** for form submissions.
- Ensure all database queries are parameterized to prevent SQL injection.
- Enforce stricter input validation on the frontend and backend.

3.2 Deliverables for Iteration 3

1. **Google OAuth2 Integration**: Fully functional Google OAuth2 login with JWT-based session management.
2. **Rate Limiting**: Rate limiting on login and OTP endpoints to protect against brute-force attacks.
3. **Security Measures**: CSRF protection and enhanced input validation on all endpoints.

3.3 Challenges to Anticipate

- Google OAuth2 integration might involve complex configuration, especially managing the callback URL and handling token exchange securely.
- Fine-tuning rate limits to balance security without negatively impacting legitimate users.

4 Iteration 4: Final Testing, Documentation, and Deployment

Objective: In **Iteration 4**, we will perform rigorous testing, document the system, and prepare it for deployment.

4.1 Tasks

1. **Comprehensive Testing**:
   - Write unit tests for major components, including OTP generation, email sending, session management, and Google OAuth2 integration.
   - Conduct end-to-end testing to ensure frontend and backend work seamlessly.
   - Perform load testing to simulate multiple users.
2. **Documentation**:
   - Update README with setup instructions, database configuration, and deployment steps.
   - Provide detailed API documentation using tools like Swagger.

- Document security enhancements implemented (e.g., rate limiting, CSRF protection).

3. **Deployment**:
   - Deploy the FastAPI application on a cloud platform (e.g., AWS, Heroku, or DigitalOcean) along with the Supabase database.
   - Configure production credentials for Gmail SMTP and Google OAuth2.
   - Prepare for the final demo showcasing signup, login (via password and Google OAuth2), and OTP verification.

4.2 Deliverables for Iteration 4

1. **Testing**: Unit tests, integration tests, and load testing results.
2. **Documentation**: Comprehensive README and detailed API documentation.
3. **Deployment**: Application deployed on a cloud platform with SSL.

4.3 Challenges to Anticipate

- Testing OAuth2 flows and load testing may require external tools or cloud services.
- Ensuring smooth deployment with secure handling of credentials.

5 Final Summary

1. **Iteration 1**: Backend and OTP system setup with FastAPI, PostgreSQL (Supabase), and email-based OTP verification.
2. **Iteration 2**: Session management using JWT and frontend-backend integration.
3. **Iteration 3**: Google OAuth2 login and advanced security features (rate limiting, CSRF).
4. **Iteration 4**: Final testing, documentation, and deployment.