INFO-SECURITY

SECURE AUTHENTICATION SYSTEM

Prepared by:

HAMMAD SIKANDAR
MASROOR REHAN
SHARJEEL NADIR
ROMAISA AHMAD
AREEBA AMJAD

19TH NOV,2024

INTRODUCTION

In an era where digital security is paramount, this project develops a secure login system that incorporates two-factor authentication (2FA) to enhance protection against unauthorized access. By integrating a password mechanism with a One-Time Password (OTP) sent to the user's registered email, the system significantly strengthens user verification processes. This dual-layer security method is crucial for thwarting potential breaches and enhancing user confidence in system integrity.

Using Python and Flask, the project combines robust backend technologies with simple front-end designs, employing SQLite for efficient data management. This approach not only safeguards against common threats like SQL injection and CSRF but also serves as a practical blueprint for secure authentication practices in web applications. Additionally, the system is designed to be scalable and adaptable, making it suitable for future enhancements and integration with other applications.

PROJECT CONTRIBUTION

HAMMAD SIKANDAR (21I-1684)

USER REGISTRATION

- Collect username, password, and email address.
- Store passwords securely using hashing techniques (e.g., bcrypt).
- Verify email address through a confirmation link.

SHARJEEL NADIR (21I-2699)

USER LOGIN AND OTP MANAGEMENT

- Authenticate users with username and password.
- Upon successful password authentication, trigger the OTP process.
- Generate a secure OTP and send it to the user's registered email.
- Set a short expiration time for the OTP (e.g., 5 minutes).
- Allow users to request a new OTP if the original expires.

ROMAISA AHMAD (21I-1702)

EMAIL INTEGRATION

- Use SMTP to send an email containing the OTP.
- Configure email server settings securely in Flask.

AREEBA (21I-1777)

SESSION MANAGEMENT

- Manage user sessions to keep the user logged in after authentication.
- Implement logout functionality.

MASROOR BIN REHAN (21I-1707)

SECURITY FEATURES

- Implement rate limiting to prevent brute-force attacks.
- Secure the application against common vulnerabilities (e.g., SQL injection, CSRF).

PROJECT COMPONENTS AND METHODOLOGIES

USER REGISTRATION - HAMMAD SIKANDAR

FUNCTIONALITY

- Users can register by providing their username, password, and email address.
- Passwords are stored securely using bcrypt hashing to ensure they cannot be compromised.
- Upon successful registration, a verification email is sent to the user containing a confirmation link to verify their email address.

TECHNOLOGIES USED

- Flask for web framework.
- bcrypt for password hashing.
- SQLite for storing user details.
- URLSafeTimedSerializer for generating secure email tokens.

METHODOLOGY

- Used bcrypt to securely hash passwords before storage.
- Employed URLSafeTimedSerializer to generate a verification link with an expiration time of 24 hours.
- Used exception handling to manage user registration errors, ensuring data integrity.

EVALUATION

- The system effectively hashed passwords and handled email verification.
- The email verification process was tested by sending the confirmation link, and it functioned as expected.
- Challenge: Ensuring the correct handling of duplicate user registrations (same username or email).

USER LOGIN AND OTP MANAGEMENT - SHARJEEL NADIR

FUNCTIONALITY

- The user logs in with a username and password.
- Upon successful authentication, the system triggers the generation of a One-Time Password (OTP).
- The OTP is sent to the registered email, with a 5-minute expiration time for security.
- Users can request a new OTP if the previous one expires or is invalid.

TECHNOLOGIES USED

- Flask for managing routes and session.
- SQLite for querying user data.
- smtplib for sending OTP emails.
- random and time for OTP generation and expiration handling.

METHODOLOGY

- Password authentication was performed using bcrypt to check if the inputted password matched the stored hashed password.
- OTP was generated using Python's random module and was stored in the session along with the timestamp for expiration handling.
- Sessions were managed using Flask's session object to track user progress through the login process.

EVALUATION

- The OTP system worked correctly, with OTPs being generated, sent via email, and verified within the time frame.
- The session management and OTP expiration logic were tested, ensuring that expired OTPs resulted in proper error handling.

EMAIL INTEGRATION - ROMAISA AHMAD

FUNCTIONALITY

- The system sends an OTP to the user’s registered email address after a successful login attempt.
- The email configuration is managed using Python's smtplib and the system securely connects to the SMTP server.

TECHNOLOGIES USED

- smtplib for email sending.
- MIMEText and MIMEMultipart for email formatting.

METHODOLOGY

- Integrated SMTP for email delivery using a Gmail server.
- Ensured that email content was securely constructed using MIME format to avoid exposure to common email vulnerabilities.
- Secured email credentials were stored in environment variables to prevent unauthorized access.

EVALUATION

- Email functionality was tested successfully for sending OTPs and verifying email addresses.
- Challenge: Ensuring that email sending did not violate Gmail’s rate limits or security policies.

SECURE ATUHENTICATION

SESSION MANAGEMENT - AREEBA

FUNCTIONALITY

- Manages user sessions once they have logged in, keeping the user authenticated for the duration of their session.
- Implements a logout function to securely terminate user sessions.

TECHNOLOGIES USED

- Flask session management.
- SQLite for session tracking.

METHODOLOGY

- Flask’s session mechanism was used to track user login states.
- The logout functionality was implemented by clearing the user session, ensuring sensitive data was wiped out upon logout.
- Secured the session data using Flask’s built-in mechanisms and secret keys.

EVALUATION

- Sessions were maintained during the login process and correctly invalidated upon logout.
- The system passed the session management tests and functioned securely with logout functionality.

SECURITY FEATURES - MASROOR BIN REHAN

FUNCTIONALITY

- Implemented rate-limiting to prevent brute-force attacks on login and OTP generation.
- Secured the application against SQL injection and Cross-Site Request Forgery (CSRF) vulnerabilities.

TECHNOLOGIES USED

- Flask-Limiter for rate limiting.
- Flask-WTF for CSRF protection.
- SQLAlchemy for secure database queries (if applicable).

METHODOLOGY

- Used Flask-Limiter to impose limits on the number of login attempts and OTP requests, preventing malicious actors from attempting brute force attacks.
- Employed Flask-WTF's CSRF protection to secure form submissions.
- Used parameterized queries with SQLite to prevent SQL injection vulnerabilities.

EVALUATION

- Rate limiting effectively blocked excessive login attempts.
- SQL injection vulnerabilities were mitigated by using parameterized queries.
- CSRF protection worked well, preventing unauthorized form submissions.

TESTING AND EVALUATION

FUNCTIONAL TESTING

- **User Registration:** Successfully tested the registration flow, ensuring data was correctly stored and emails sent.
- **Login & OTP:** Verified OTP functionality, ensuring OTP generation, expiration, and re-generation worked as expected.
- **Email Integration:** Tested email sending, ensuring OTPs were delivered without delay or error.
- **Session Management:** Tested user sessions and logout functionality, ensuring users were properly logged out after their session expired.

SECURITY TESTING

- Conducted penetration testing to ensure protection against brute-force login attacks, SQL injection, and CSRF.
- The rate-limiting and CSRF protection features were evaluated for effectiveness against common attacks.

CONCLUSION

The secure authentication system developed for this project successfully integrates multiple security measures, including secure password storage, email-based verification, OTP management, and session handling. The collaborative effort of the team members led to a well-structured and secure application. The project demonstrated the importance of implementing security features to safeguard user data and prevent common attack vectors in web applications.

FUTURE RECOMMENDATIONS

- Explore implementing multi-factor authentication (MFA) beyond email OTPs, such as using authentication apps (e.g., Google Authenticator).
- Further optimize rate limiting and session management to handle higher traffic securely.

SECURE AUTHENTICATION FRONT-END

SCREENSHOTS