

The background features a vibrant gradient from purple to pink. In the top left, there are two small, faceted geometric shapes (one orange, one blue) and a small orange equals sign. In the top right, a larger, faceted geometric shape is surrounded by a yellow curved line and a small blue sphere. In the bottom right, there are two more small blue spheres. The main title 'Exercise Work' is centered in a large, white, sans-serif font.

Exercise Work

COMP.SEC.300-2024-2025-1 – Secure Programming
By: Muhammad Hasan Usama



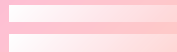
Introduction

Project Overview:

- The web application is a user authentication system with role-based access control (admin, analyst, user).
- The app is designed to demonstrate secure programming practices following OWASP guidelines.

Key Features:

- Signup/Login functionality
- Role management (admin/analyst/user)
- Secure session management
- Password hashing for security
- Database interaction with SQLite





Overview of OWASP Guidelines Applied



OWASP (Open Web Application Security Project) focuses on common web application vulnerabilities.

Key OWASP Principles Applied:

- Injection Prevention (SQL injection)
- Secure Authentication (password storage)
- Session Management (secure cookies)
- Input Validation and Output Encoding (XSS)

Goal: Mitigate common security vulnerabilities like SQL injection, XSS, weak password storage, and insecure session handling.





What My Code Does

Login Route (/login):

- Secure Authentication: The app checks credentials, hashes the password securely, and uses sessions to store the user's role.
- Role-Based Redirect: After login, the user is redirected based on their role (admin, analyst, or user).

Signup Route (/signup):

- Input Validation: The form checks the username, email, and password with regular expressions to prevent invalid input.
- Password Hashing: Passwords are securely hashed using `werkzeug.security` before storing in the database.
- Error Handling: If the username or email already exists, the user is notified.

Admin Panel (/admin):

- Role Management: Admins can change the roles of users (e.g., give analyst privileges).
- Access Control: Only users with the "admin" role can access this page.

Session Management:

- Secure Cookies: Sessions are managed with secure cookies (`SESSION_COOKIE_SECURE`, `SESSION_COOKIE_HTTPONLY`).
- Prevent Session Fixation: The session is cleared upon login and logout.





How I Applied OWASP Guidelines

SQL Injection Prevention:

What I Applied: I used parameterized queries (e.g., `cursor.execute("SELECT ... WHERE username = ?", (username,))`) to safely interact with the database.

OWASP Principle: Prevents attackers from injecting malicious SQL code into the application.

Secure Password Storage:

What I Applied: Passwords are hashed using `bcrypt` via `generate_password_hash` and stored in the database.

OWASP Principle: Ensures that even if the database is compromised, the passwords cannot be easily decrypted.

Session Management:

What I Applied: Secure session cookies with `SESSION_COOKIE_HTTPONLY`, `SESSION_COOKIE_SECURE`, `SESSION_COOKIE_SAMESITE` settings.

OWASP Principle: These settings ensure that session cookies are protected from client-side access and only work over HTTPS, helping prevent session hijacking and XSS attacks.

Input Validation:

What I Applied: Regex validation for user inputs (username, email, and password) ensures only valid inputs are processed.

OWASP Principle: Helps prevent malicious or malformed input that could lead to security vulnerabilities (e.g., XSS).





Analyst Dashboard – Cryptography Integration



Key Points

- **Encryption & Secure Storage:** Data is encrypted using cryptographic algorithms before being saved in the database.
- **Partial Decryption for Analysis:** The analyst role can access and analyze partially decrypted data for insights.
- **Role-Based Access Control:** Only users with the analyst role can decrypt and view this data.
- **First-Time Integration:** First-time integration of cryptographic data handling into the frontend system for secure analysis by me.





What Needs Improvement (OWASP Focused)



Second Program: Admin Creation

- What it Does: The second program adds an admin user to the database.
- Security Issue: The admin credentials are hardcoded into the script, which is risky as sensitive data can be exposed.

OWASP Best Practice: Replace hardcoded credentials with environment variables or secure vaults to store sensitive data, ensuring they're not exposed in source code.

SQL Injection Prevention in the Second Program:

What I Applied: Though no user input is taken in the second program, I would still ensure that any future database interaction uses parameterized queries, even for administrative tasks.

OWASP Best Practice: Always use parameterized queries to avoid any potential risks related to SQL injection.



Conclusion



Key Takeaways:

- My Flask application follows OWASP best practices for web application security.
- SQL injection prevention, secure password storage, and session management are properly handled.
- The admin creation script needs improvements, such as replacing hardcoded credentials with environment variables.

Final Thought:

Following OWASP guidelines in web development not only enhances security but builds trust and reliability for users interacting with the application.





THANKS!



CREDITS: This presentation template was created by
Slidesgo, including icons by **Flaticon**, infographics &
images by **Freepik**