DAR ES SALAAM INSTITUTE OF TECHNOLOGY

GROUP ASSIGNIMENT

MODULE NAME: WEB APPLICATION AND DEVELOPMENT

TITLE: COU 07503

CLASS : BENG21-COE 1

GROUP MEMBERS

S/N	NAME	REGISTRATION NUMBER	REMARK
1	FRANK URASSA	2102302218633	
2	MIRIAM ROBERT	2102302121423	
3	FAUDHIA MSOKE	2102302117140	
4	BULABO PAULO	2102302226933	
5	DANIEL MSEMAKWELI	2102302228616	

TO DO LIST APPLICATION

SECURITY TEST

Performing a comprehensive security testing on a Django to do list web application is crucial to identify and address potential vulnerabilities. Below, are the strategies we performed on the application we developed including information on vulnerabilities, their sources, and solutions we implemented and suggest to conduct.

Security Testing and Vulnerabilities

a. Cross-Site Scripting (XSS):

Description: XSS occurs when an attacker injects malicious scripts into web pages, which are then executed by the user's browser.

Testing Approach:

Input validation: Check if user inputs are properly sanitized.

Output encoding: Ensure that data displayed in HTML is properly encoded.

Content Security Policy (CSP): Implement a strict CSP to control script sources.

Solutions:

Sanitize input data using Django forms and validators.

Use Django's `mark_safe` cautiously for rendering HTML content.

Implement Content Security Policy headers in your application.

b. SQL Injection:

Description: SQL injection involves injecting malicious SQL queries through user inputs, potentially leading to unauthorized database access.

Testing Approach:

Use parameterized queries or Django's ORM for database interactions.

Input validation: Validate and sanitize user inputs to prevent malicious SQL injection attempts.

Solutions:

Prefer Django's Object-Relational Mapping (ORM) for database queries.

Use parameterized queries to avoid direct insertion of user input into SQL statements.

Implement proper input validation and sanitization using Django forms.

c. Penetration Testing:

Description: Penetration testing involves simulating real-world attacks to identify vulnerabilities and weaknesses.

Testing Approach:

Conduct automated vulnerability scans.

Perform manual penetration testing to identify potential weaknesses.

Test for common security misconfigurations.

Solutions:

Regularly conduct penetration testing to identify and fix vulnerabilities.

Keep the system and third-party libraries up-to-date.

Implement a robust web application firewall (WAF) to mitigate potential threats.

Vulnerability Sources

a. Cross-Site Scripting (XSS):

Sources:

Lack of input validation and sanitation.

Improper use of `mark_safe`.

Absence or misconfiguration of Content Security Policy.

b. SQL Injection:

Sources:

Direct concatenation of user input into SQL queries.

Lack of parameterized queries or ORM usage.

Insufficient input validation and sanitation.

c. Penetration Testing:

Sources:

Outdated software and libraries.

Misconfigurations in web servers or Django settings.

Insufficient protection against common vulnerabilities.

Solutions on Vulnerabilities

a. Cross-Site Scripting (XSS):

Solutions:

Implement proper input validation using Django forms and validators.

Use Django's ORM and parameterized queries.

Encode output using Django templates.

Configure a strict Content Security Policy.

b. SQL Injection:

Solutions:

Use Django's ORM for database interactions.

Employ parameterized queries to avoid direct insertion of user input.

Implement rigorous input validation and sanitation using Django forms.

c. Penetration Testing:

Solutions:

Regularly update the system and third-party libraries.

Conduct periodic penetration testing to identify and address vulnerabilities.

Configure web application firewalls to filter and block potential threats.

Conclusion

By following these solutions, you can enhance the security of our Django web application and mitigate the identified vulnerabilities. Regularly updating and testing the application's security measures will help maintain a robust defense against evolving threats.