Introduction:

This report aims to identify and describe five security flaws present in a web application and provide specific fixes for each flaw. The web application has been evaluated for common security vulnerabilities based on the OWASP Top Ten list. Each flaw will be presented along with its description, potential risks, and practical steps to mitigate the vulnerabilities.

------------------------------------------------

FLAW 1: Cross-Site Scripting (XSS)

Source Link: detail.html, Line 7

Description:
Cross-Site Scripting (XSS) is a vulnerability that allows an attacker to inject malicious scripts into a web application. In the provided code, the polls/detail.html template does not properly escape user inputs when displaying question choices. As a result, it leaves the application exposed to potential XSS attacks.

Potential Risks:
If an attacker successfully injects malicious scripts into the page, they can steal sensitive user data, such as session cookies or login credentials, and execute unauthorized actions on behalf of the user.

Fix:
To mitigate the XSS vulnerability, proper input validation and output encoding are required. Modify the polls/detail.html template to escape user inputs using the safe filter.

------------------------------------------------

FLAW 2: Insecure Direct Object References (IDOR)

Source Link: views.py, Line 24

Description:
Insecure Direct Object References (IDOR) occur when an application exposes sensitive information or functionality by referencing objects directly via user-supplied input, such as IDs. In the provided code, the vote function lacks proper access control, allowing any user to vote on any question without permission checks.

Potential Risks:
This vulnerability could lead to unauthorized access to sensitive resources, manipulation of data, and unauthorized actions within the application.

Fix:
To address IDOR, implement proper permission checks to ensure that only authenticated users can vote on questions. Add the login_required decorator to the vote function to enforce authentication.

------------------------------------------------

FLAW 3: Security Misconfiguration (DEBUG = False)

Source Link: settings.py, Line 28

Description:
Security misconfiguration occurs when a web application is deployed with insecure settings. In the provided code, the application's DEBUG mode is set to True, which is unsuitable for a production environment.

Potential Risks:
In a production environment, having DEBUG mode enabled could expose sensitive information, such as stack traces and database credentials, to attackers.

Fix:
In settings.py, set DEBUG to False in a production environment to prevent detailed error information from being displayed.

------------------------------------------------

FLAW 4: Cross-Site Request Forgery (CSRF)

Source Link: settings.py, Line 48

Description:
Although not explicitly listed in the OWASP Top Ten 2017, Cross-Site Request Forgery (CSRF) is still a security concern. The application is not protected against CSRF attacks.

Potential Risks:
CSRF attacks can lead to unauthorized actions being executed on behalf of authenticated users.

Fix:
Django provides built-in CSRF protection using a middleware (CsrfViewMiddleware). It is automatically included in the MIDDLEWARE setting in settings.py. Ensure that the CSRF token is present in your forms when submitting data to the server.

------------------------------------------------

FLAW 5: SQL Injection

Source Link: views.py, Line 18

Description:
Although the application is using SQLite, which is not vulnerable to classic SQL injection, it's crucial to demonstrate how to handle user input safely, especially when working with other databases.

Potential Risks:
Without proper input validation, the application could be susceptible to SQL injection when working with vulnerable databases.

Fix:
Implement Django's query parameterization to handle user input securely and prevent SQL injection attacks. Modify the vote function to use query parameterization.

------------------------------------------------

Conclusion:

This report has identified and described five security flaws in the web application based on the OWASP Top Ten 2017 list. Each flaw was accompanied by a detailed description of the vulnerability, potential risks, and specific code fixes to address the issues. By following the provided fixes and implementing secure coding practices, the web application can be better safeguarded against common security threats.

It's essential to prioritize web application security throughout the development process, regularly perform security testing, and stay informed about emerging threats and best practices to ensure a robust and secure web application.