TASK2:

Objective:

The objective of this task was to learn about common web application vulnerabilities by analyzing a simple web application using OWASP ZAP. This involved identifying and understanding vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

Tools Used:

OWASP ZAP: A popular open-source web application security scanner.

WebGoat: A deliberately vulnerable web application used for security training.

Vulnerability Analysis:

1. SQL Injection:

- **Discovery**: While testing the login functionality on WebGoat, it was observed that the application concatenated user input directly into the SQL query without proper sanitization.
- **Exploitation**: By entering a malicious SQL query (e.g., ' OR 1=1; --), bypassing authentication and gaining unauthorized access to the application was possible.
- **Danger**: SQL injection can lead to data theft, unauthorized access, and even system compromise.
- **Mitigation**: Input validation, parameterized queries, and prepared statements should be used to prevent SQL injection.

2. Cross-Site Scripting (XSS):

- **Discovery**: A vulnerable search functionality was identified where user input was reflected back in the search results without proper encoding.
- **Exploitation**: By injecting malicious JavaScript code (e.g., <script>alert('XSS attack!');</script>) into the search query, it was possible to execute arbitrary code in the victim's browser.
- **Danger**: XSS can be used to steal sensitive information, redirect users to malicious websites, and execute malicious code.

Mitigation: Output encoding, input validation, and using a Content Security Policy (CSP) can help prevent XSS.

3. Cross-Site Request Forgery (CSRF):

- **Discovery**: A vulnerable "Change Password" functionality was found that lacked CSRF protection.

- **Exploitation**: By crafting a malicious link that triggered the "Change Password" action without the user's knowledge or consent, it was possible to modify their password.
- **Danger**: CSRF can be used to perform unauthorized actions on behalf of a logged-in user, such as making purchases, transferring funds, or deleting data.
- **Mitigation**: Using a CSRF token and verifying it on the server-side can prevent CSRF attacks.

Conclusion:

This task provided valuable insights into common web application vulnerabilities and how they can be exploited. By understanding these vulnerabilities and implementing appropriate mitigation techniques, developers can significantly improve the security of their web applications.