**Task 2: Introduction to Web Application SecuritySetup:**

git clone [git@github.com:WebGoat/WebGoat.git](mailto:git@github.com:WebGoat/WebGoat.git)

**Step 2**: Ensure you have **Java** installed on your machine

java -jar webgoat-server-x.x.x.jar

step3:

run localhost 🡪http://localhost:8080/WebGoat

1. **Perform Basic Vulnerability Analysis:**

In OWASP ZAP, perform an **active scan** on the WebGoat instance. ZAP will identify vulnerabilities automatically, including:

**SQL Injection (SQLi)**

**Cross-Site Scripting (XSS)**

**Cross-Site Request Forgery (CSRF)**

1. **Explore Vulnerabilities:**

**SQL Injection**:

Find a vulnerable form (e.g., login or search form) where user input is not sanitized.

Attempt to exploit SQL injection by inserting common payloads like:

‘ union 1=1 –

**Cross-Site Scripting (XSS)**:

* Locate a vulnerable field that echoes user input back into the page without proper escaping.
* Exploit XSS by injecting a simple script, like:

<script>alert(‘1’)</script>

**Cross-Site Request Forgery (CSRF)**:

* Check for forms that perform actions (e.g., changing user settings) without validating if the request originated from an authorized user.
* Attempt to exploit CSRF by creating a malicious form on another site, forcing the user to submit a request without their knowledge.

<form action="http://localhost:8080/WebGoat/vulnerabilities/csrf" method="POST">

<input type="hidden" name="action" value="changePassword">

<input type="submit" value="Submit">

</form>

**Tools Overview:**

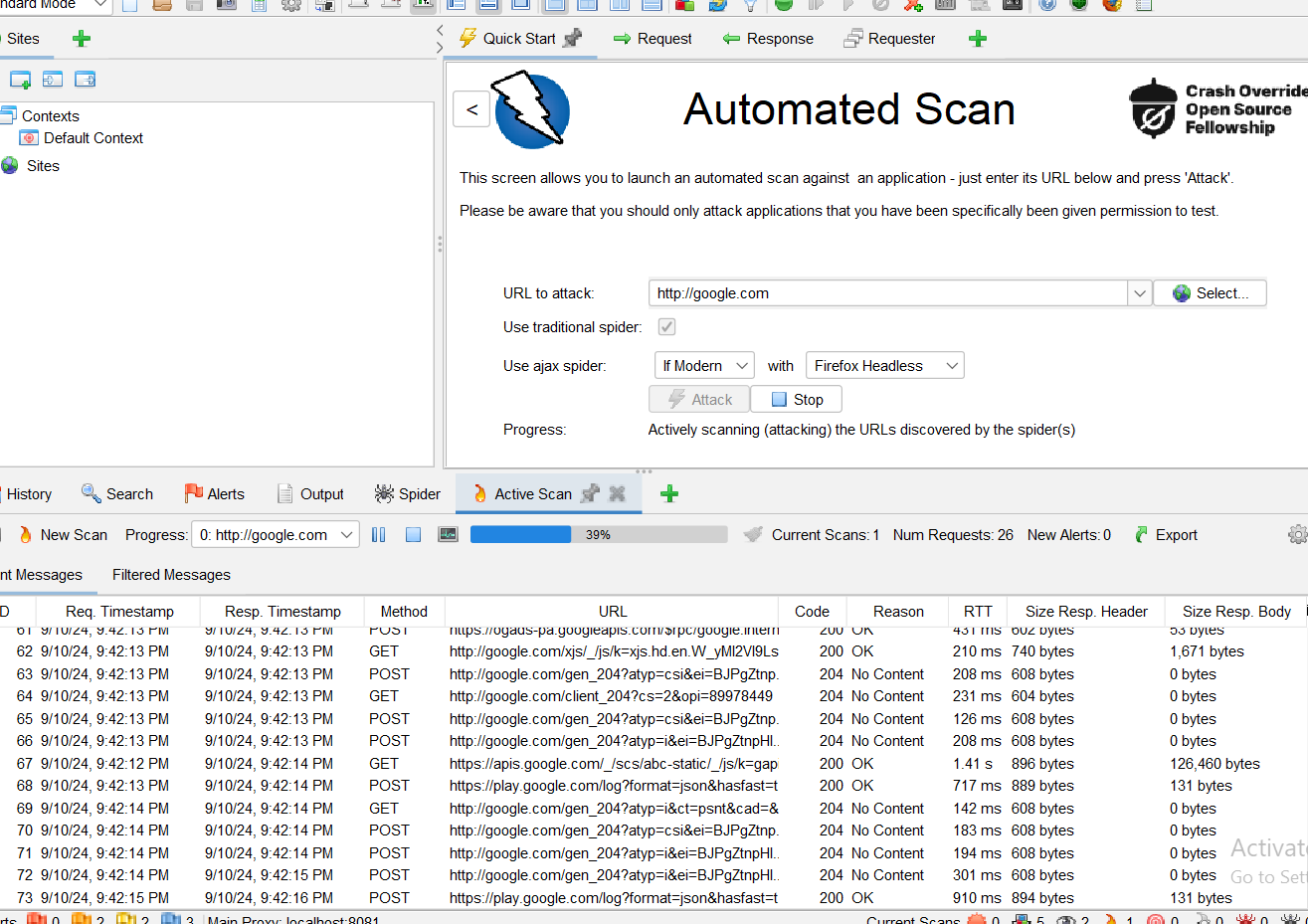
* **WebGoat**: A vulnerable web application used for learning and practicing security exploits.
* **OWASP ZAP**: A tool for automated vulnerability scanning and exploitation, commonly used in penetration testing.

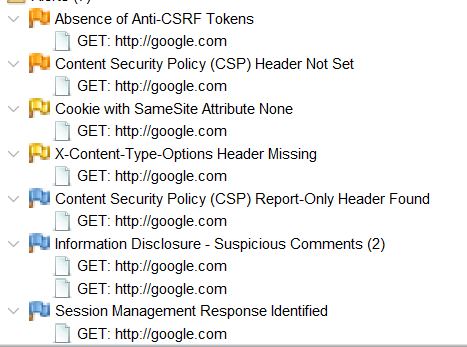
**Vulnerabilities**:

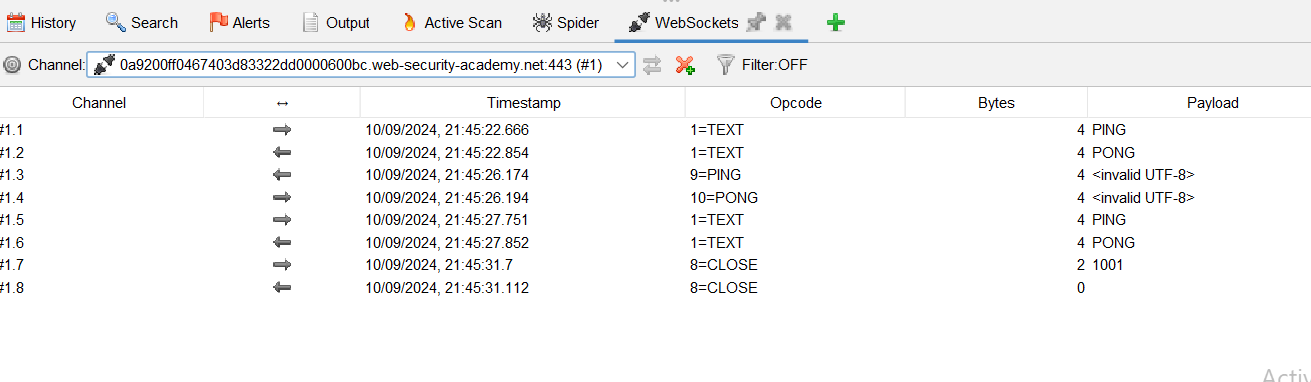
* **SQL Injection**:
  + **Discovery**: Found through scanning and manual testing of input fields.
  + **Danger**: Can lead to unauthorized data access or manipulation.
* **Cross-Site Scripting (XSS)**:
  + **Discovery**: Identified in fields reflecting user input.
  + **Danger**: Can allow attackers to execute scripts in users' browsers.
* **Cross-Site Request Forgery (CSRF)**:
  + **Discovery**: Detected in forms lacking anti-CSRF tokens.
  + **Danger**: Can make unauthorized changes on behalf of authenticated users.

**Mitigation Suggestions**:

* **SQL Injection**: Use prepared statements and parameterized queries.
* **XSS**: Sanitize and escape user inputs, and use content security policies (CSP).
* **CSRF**: Implement anti-CSRF tokens in forms and verify requests.







Find sql injection attack

Protswigger labs:

Sql injection: https://github.com/nadamohamedabdelrahman/portswigger\_labs/blob/main/SQl%20injection.docx

csrf: https://github.com/nadamohamedabdelrahman/portswigger\_labs/blob/main/Csrf.docx

xss: https://github.com/nadamohamedabdelrahman/portswigger\_labs/blob/main/Cross-site%20scripting.docx