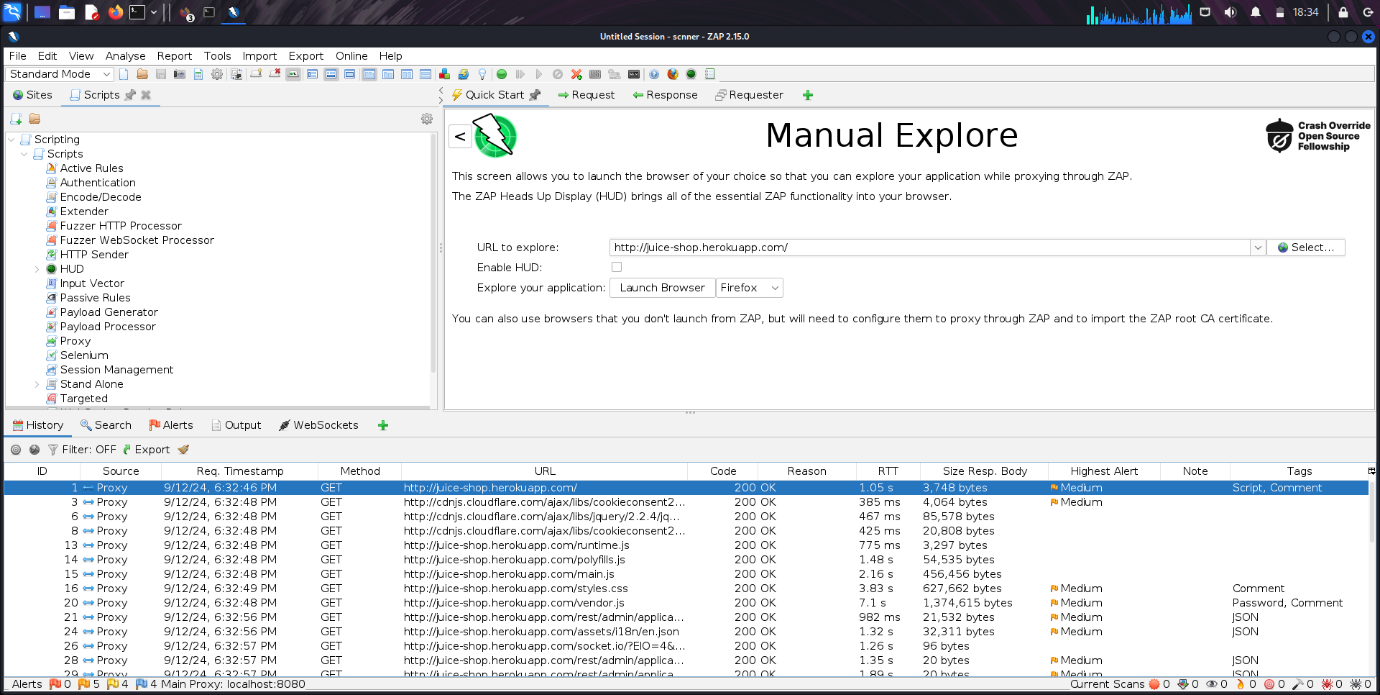
**Task 2: Introduction to Web Application Security (Solution)**

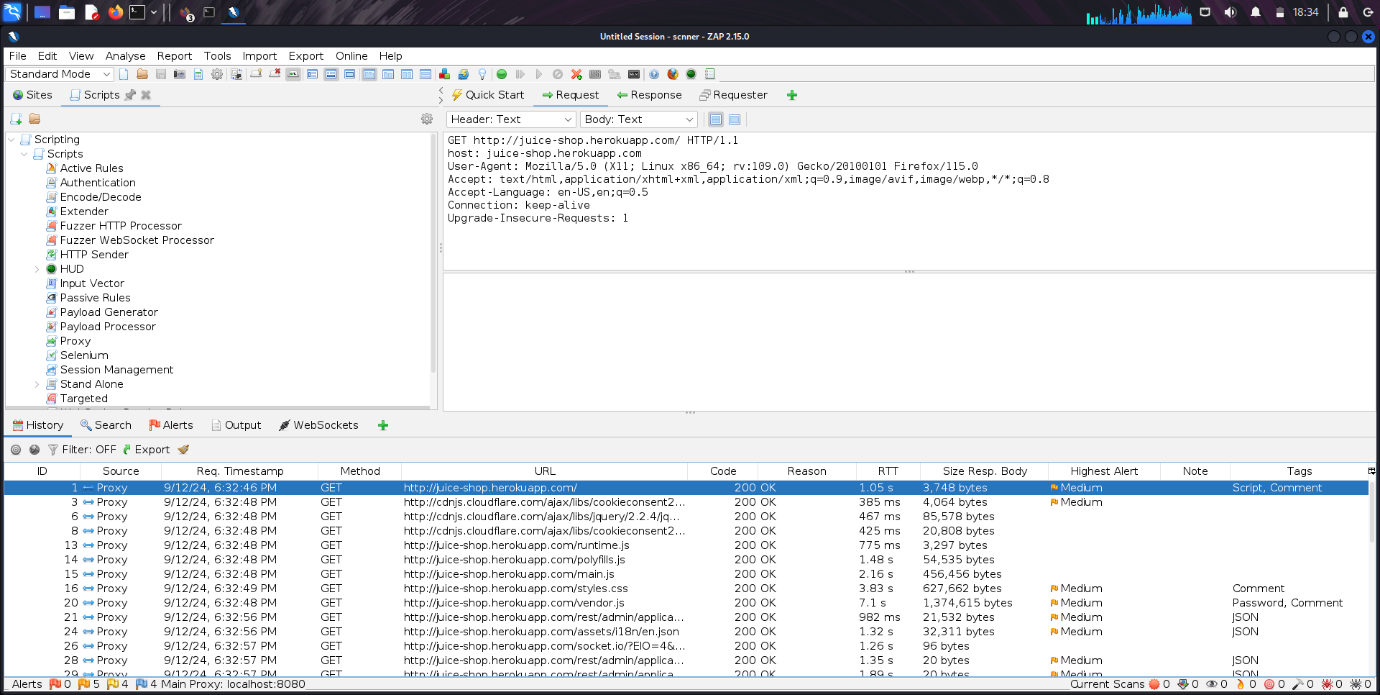
**Objective**

This report aims to introduce common web application vulnerabilities through a hands-on analysis of a vulnerable web application, Web Goat and OWASP juice shop. By identifying and understanding vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF), we can gain insights into how attackers exploit weaknesses in web applications.

**Methodology**

1. **Setup:**
   * Installed and configured web goat and OWASP juice shop on a local machine.
   * Familiarized myself with the application's functionalities and access points.
2. **Basic Vulnerability Analysis:**
   * Utilized OWASP ZAP to scan Web Goat and OWASP juice shop for potential vulnerabilities.
   * Focused on identifying instances of SQL Injection, XSS, and CSRF.
3. **Vulnerability Exploration:**
   * Delved into the details of each identified vulnerability using OWASP ZAP's descriptions.
   * Attempted to manually exploit these vulnerabilities using basic techniques (e.g., injecting SQL code into login forms).
   * Documented the vulnerabilities, including discovery methods, potential risks, and mitigation strategies.

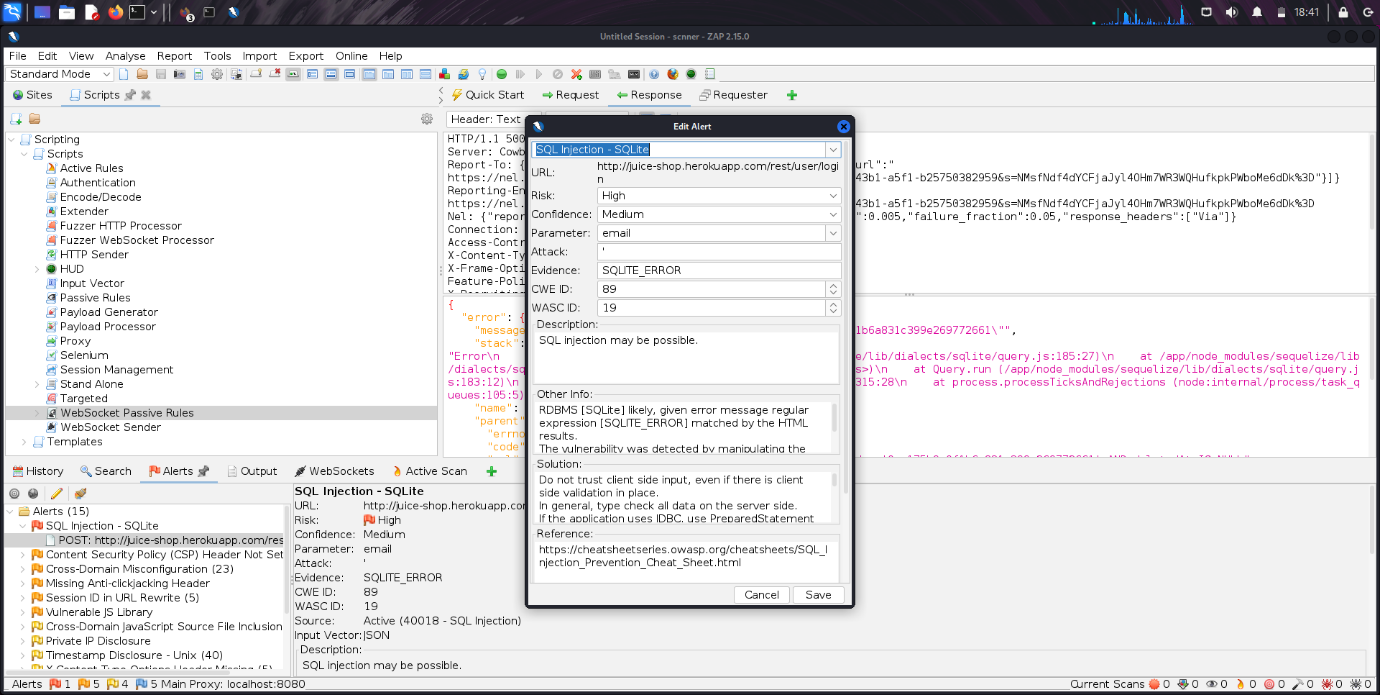


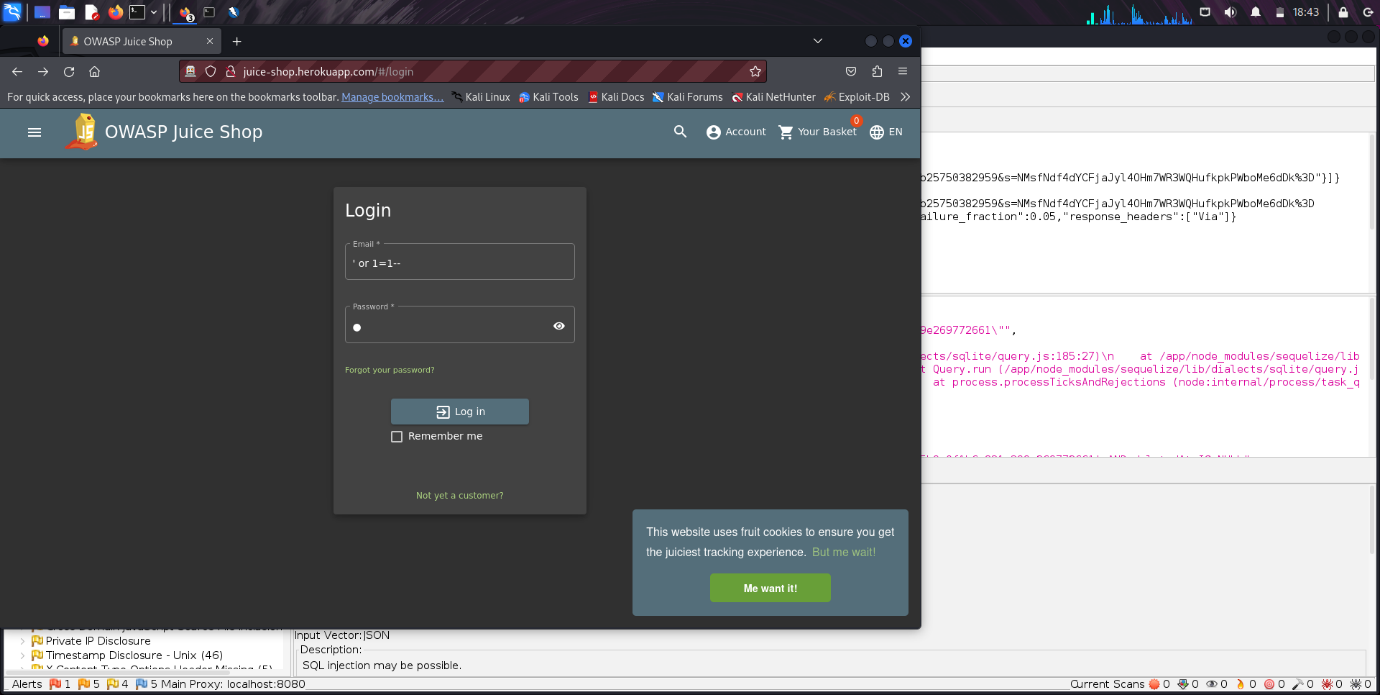
****

**Findings and Analysis**

**1. SQL Injection**

* **Discovery:** While attempting to log in with incorrect credentials, I noticed that the application returned an error message revealing the underlying SQL query. This indicated a potential SQL Injection vulnerability.
* **Exploitation:** By injecting malicious SQL code into the username field, I was able to execute arbitrary SQL commands, such as retrieving sensitive information from the database.
* **Risk:** SQL Injection can lead to unauthorized access to sensitive data, database manipulation, and even system compromise.

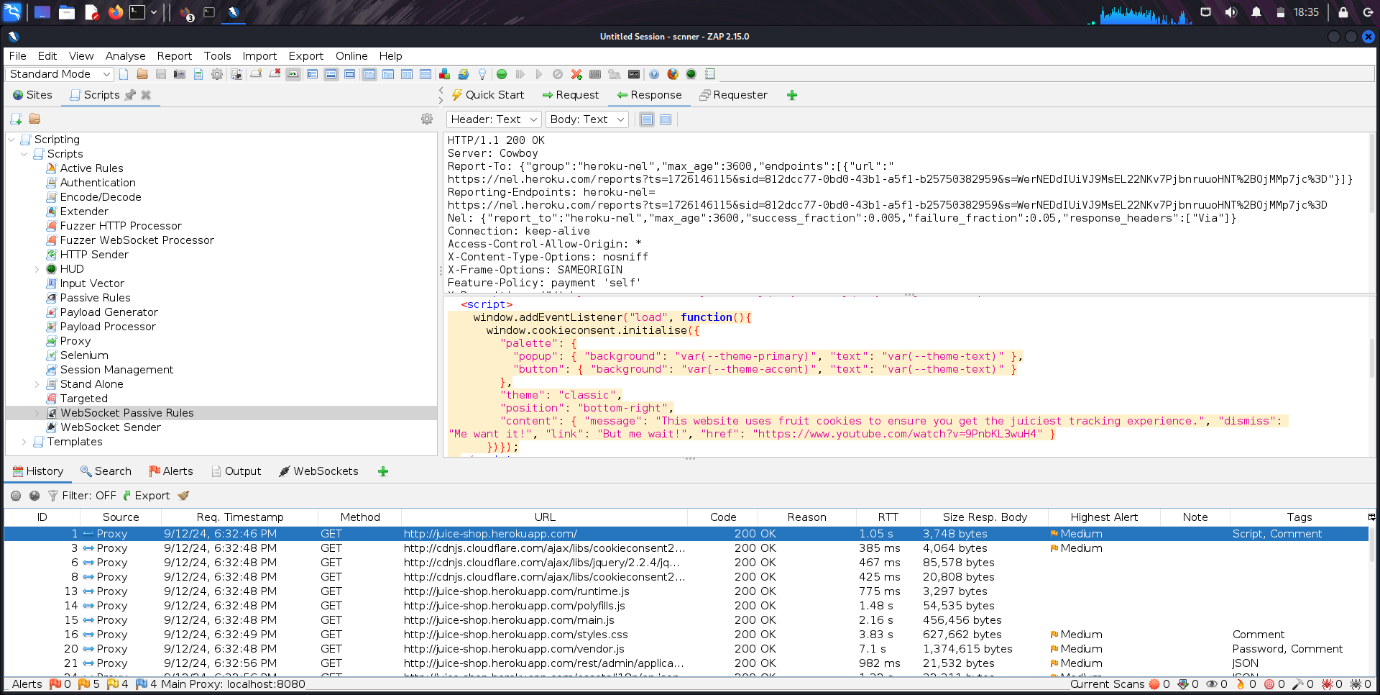




* **Mitigation:**
  + Input validation: Sanitize and validate user-provided input to prevent malicious code injection.
  + Prepared statements: Use parameterized queries to separate data from SQL code, preventing direct execution of user-supplied input.

**2. Cross-Site Scripting (XSS)**

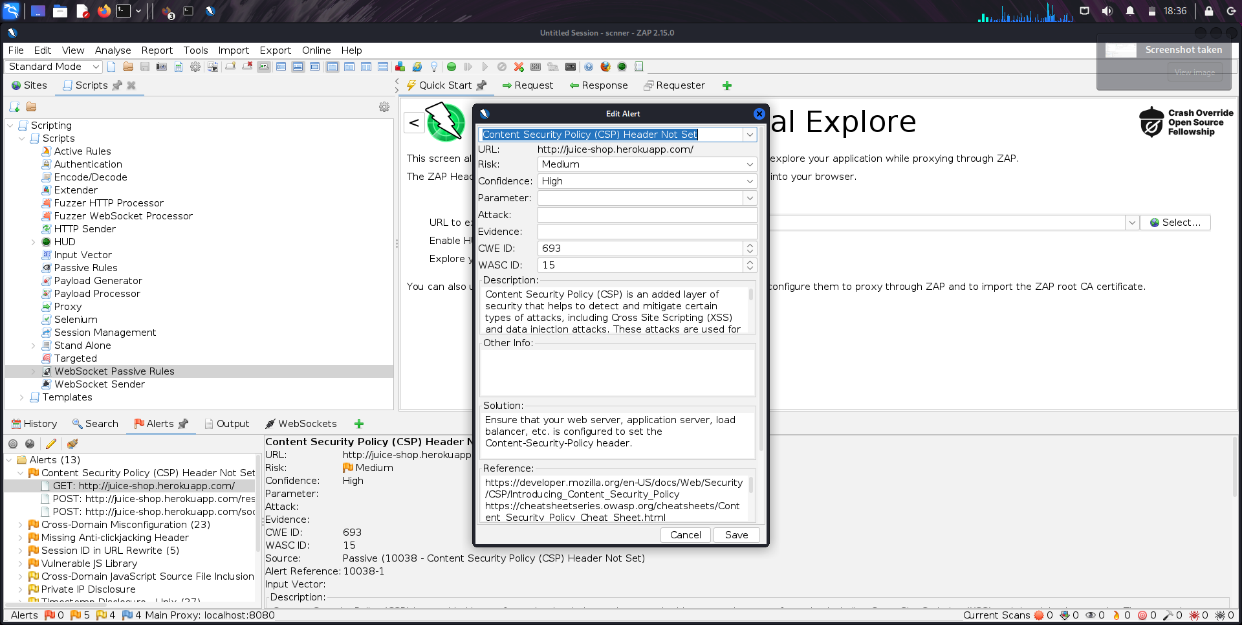
* **Discovery:** When entering malicious script code into a text field on the application, the script was executed on the client-side, potentially compromising user sessions or executing arbitrary code.
* **Exploitation:** By injecting a JavaScript alert box into a comment field, I was able to execute arbitrary JavaScript code in the user's browser.
* **Risk:** XSS can lead to unauthorized access to user accounts, data theft, and phishing attacks.

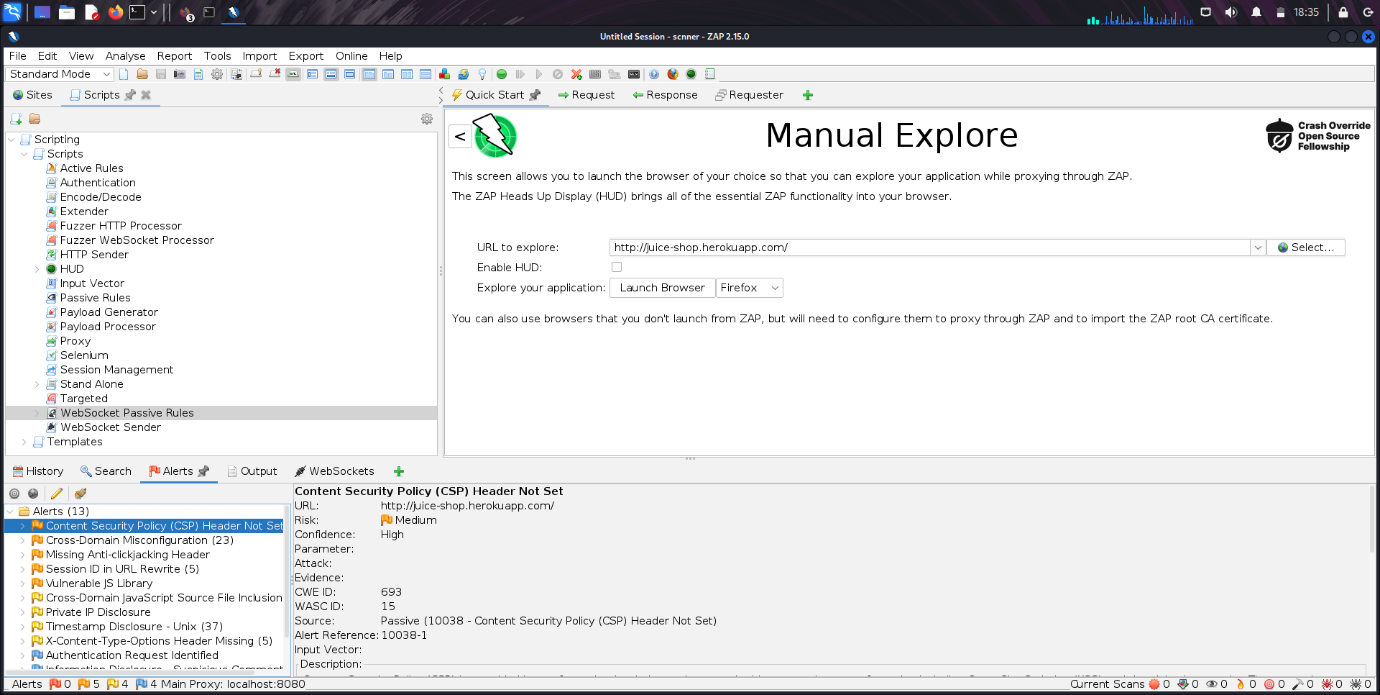


* **Mitigation:**
  + Output encoding: Encode all user-generated content before outputting it to the browser to prevent malicious script execution.
  + Content Security Policy (CSP): Use CSP to restrict the execution of scripts from untrusted sources.

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

* **Discovery:** By crafting a malicious link that triggered a sensitive action on the application (e.g., changing a password), I was able to force an authenticated user to perform the action without their knowledge.
* **Exploitation:** A user who clicked on the malicious link would unknowingly execute the sensitive action, potentially compromising their account.
* **Risk:** CSRF can lead to unauthorized actions on behalf of authenticated users, such as password changes or unauthorized transactions.





* **Mitigation:**
  + Synchronizer Token Pattern (STP): Use a unique token to verify the authenticity of requests.
  + Double Submit Cookie: Use a unique token in both the request and the cookie to prevent CSRF attacks.

**Conclusion**

Through this hands-on analysis of Web Goat and OWASP juice shop, we have gained valuable insights into common web application vulnerabilities, including SQL Injection, Cross-Site Scripting, and Cross-Site Request Forgery. By understanding how these vulnerabilities are exploited and implementing appropriate mitigation strategies, developers can significantly improve the security of their web applications.