Database and Web Security Challenges and Solutions

By

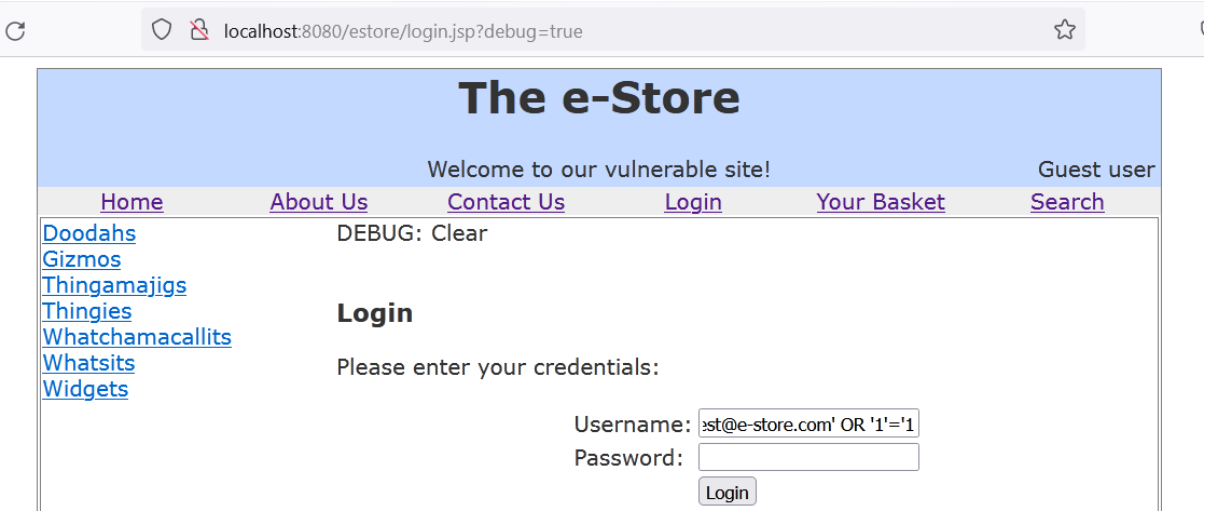
Usiwoghene Ekebor

Table of Content **Page Number**

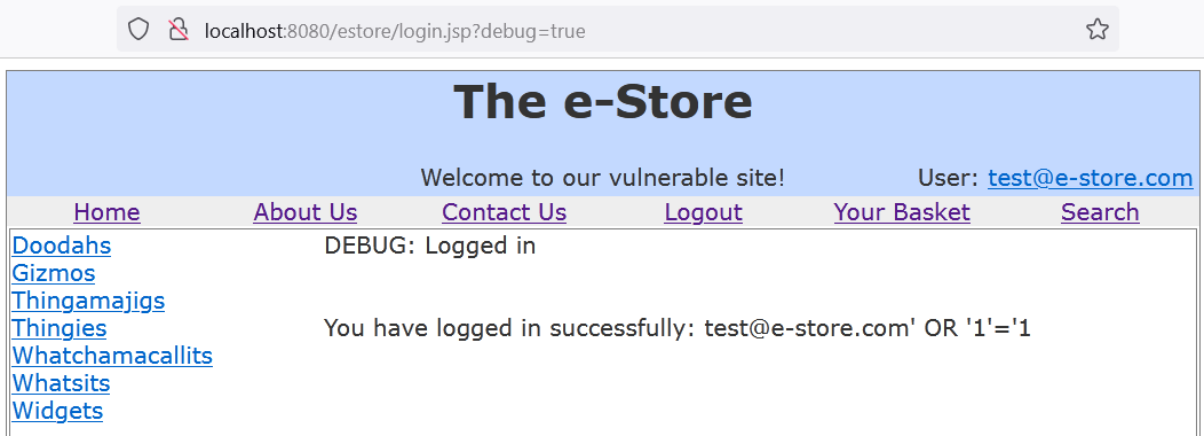
1.0 Task 1: Offensive Security

1.1 Challenge 1 to 3

From the login page attempts were made to try and use test@e-store.com to login but few modifications were made to the username, by adding **'OR'1'='1** which is an SQL injection code. The username **test@e-store.com' OR '1'='1** was used without password as shown below in the screenshots and then login was clicked.



When the Login link is clicked, access to the webstore is successful as shown below in the screenshot



To explain, the ' OR '1'='1 SQL injection code, the ' added after the email indicates and end to that statement, the OR specify that one of either the username or 1=1 needs to be true, and sure enough 1=1 is always true. Refreshing the score page shows that the challenge has turned green. This step is repeated with the username (user1@e-store.com ' OR '1'='1 and admin@e-store.com ' OR '1'='1) in challenge 2 and 3.

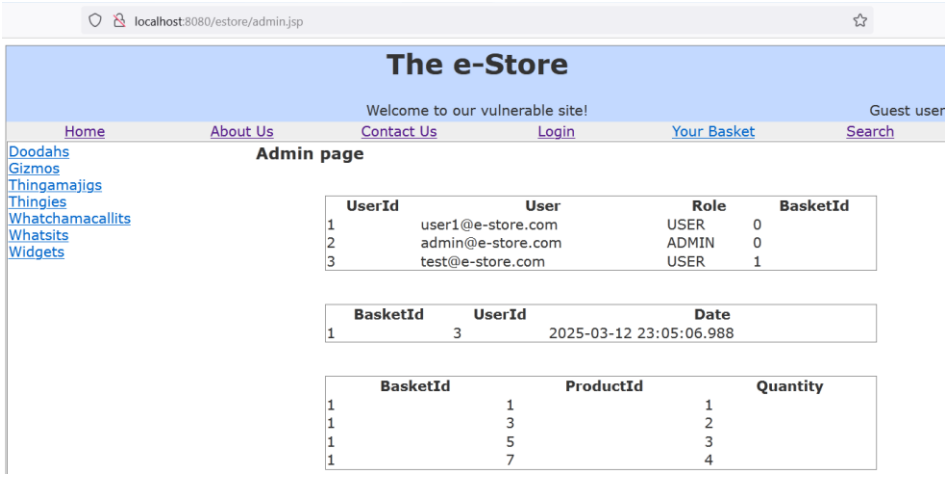
1.2 Challenge 4: Find hidden content as a non admin user

To approach this challenge the first thing that was done from the home page of the e-store website, was to right click on any part of the page and select the 'view page source' which opened in a new tab, from this page the html source code that shows the home, about, contact pages of e-store website. Furthermore, a code with the html comment tag showing the admin page can be

seen from line 39 the code screenshot below. This reveals that contents to the admin page could potentially be accessed.

```
32 </td>
33 </tr>
34 <tr>
35 <td align="center" width="16%" BGCOLOR=#EEEEEE><a href="home.jsp">Home</a></td>
36 <td align="center" width="16%" BGCOLOR=#EEEEEE><a href="about.jsp">About Us</a></td>
37
38 <td align="center" width="16%" BGCOLOR=#EEEEEE><a href="contact.jsp">Contact Us</a></td>
39 <!-- td align="center" width="16%"><a href="admin.jsp">Admin</a></td-->
40
41 <td align="center" width="16%" BGCOLOR=#EEEEEE>
42
43         <a href="login.jsp">Login</a>
```

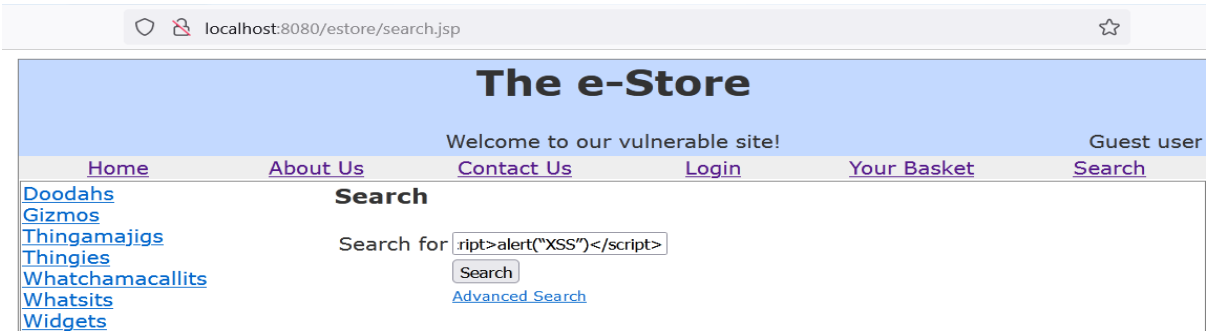
Going back to the home page of the e-store, from the address bar, **home.jsp** was replaced with **admin.jsp** and this granted immediate access to admin information even without logging to the web app.



Navigating to the score page, this task has turned to green which shows it has been completed. This vulnerability can be categorised as Security Misconfiguration from the OWASP top 10.

1.3 Challenge 6: Level 1: Display a popup using: <script>alert("XSS")</script>

Cross site scripting is a form of security vulnerability found in some web app or site which allows attackers to inject malicious scripts in the web app or site. To test and exploit this type of vulnerability on the e-store webapp, any available text boxes was explored, in this case, the search box in the search page was used and this script **“<script>alert("XSS")</script>”** was inserted as shown on the screenshot below.

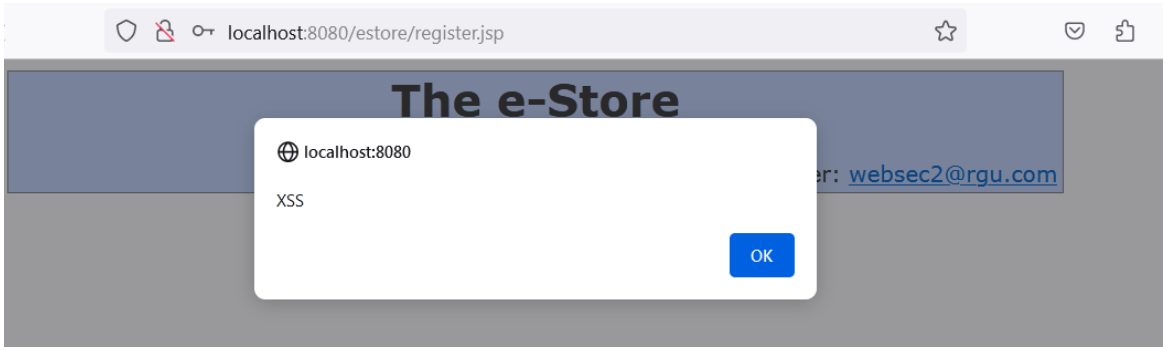


The result of the injected code is the pop-up display of XSS as shown in the screenshot below

1.4 Challenge 7: Level 2: Display a popup using: <script>alert("XSS")</script>

Displaying another popup with the same script, another page where text can be can inputted was explored, this time the login page. This challenge is a little more tricker than the previous one, from the login page we first try to register a new user and add html code line in the username. We use the email websec@rgu.comHello as username, and hack1 as password and the registration was successful. This means the e-store web app is executing the html script. Now that this is known, returned to the login page and created another user, but this time add the XSS script at the front of the username (websec2@rgu.com<script>alert("XSS")</script>) and password hack2.

When register is clicked, this displays the popup with XSS message as shown in the screenshots, and when we click on ok we noticed that the registration was successful.

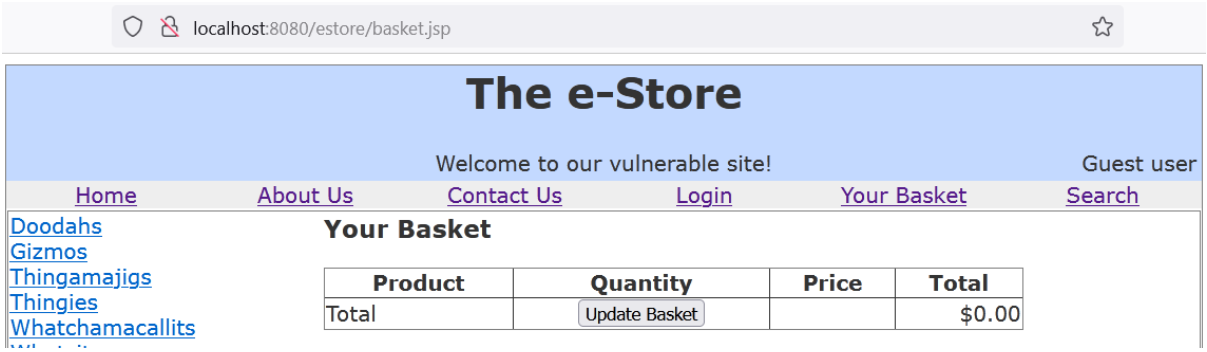


Subsequently, any page the user visit while logged in, the popup message will keep displaying which means it's a stored XSS. When the score page is checked, this challenge has turned green.

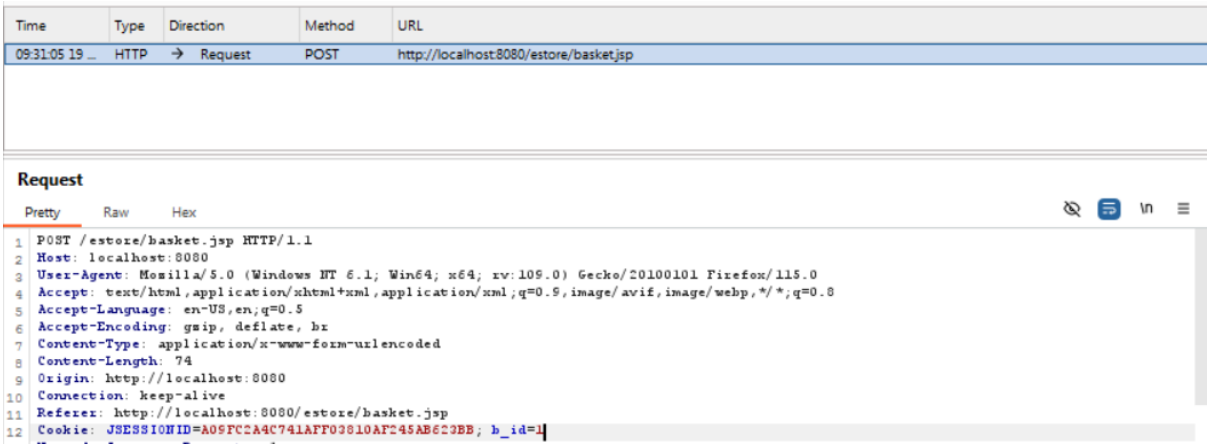
1.5 Challenge 8: Access someone elses basket

In this challenge, access to an existing user basket as non-user was achieved and the first thing to do is visit the admin page to see the different existing baskets and their IDs as shown in the screenshot. Note the ID of all existing account.

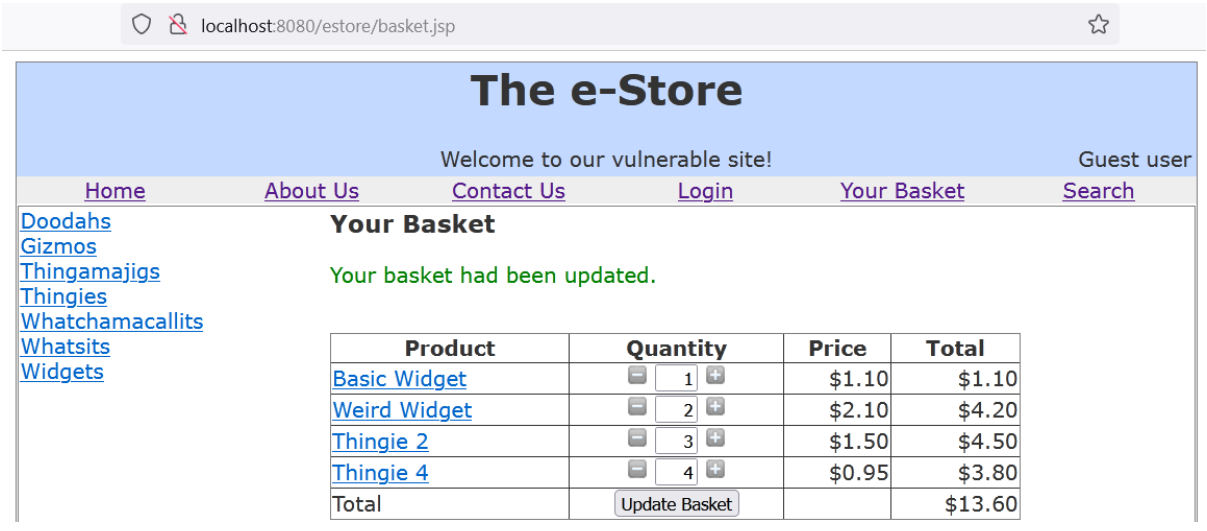
Then run Burp and configure to capture traffic from the browser, then from visiting the basket page, update basket was clicked as seen on the screenshot below.



The traffic was captured at Burp, as seen from the screenshot in line 12, there is a b_id, we set that to 1 which is the basket ID for test@e-store.com account.

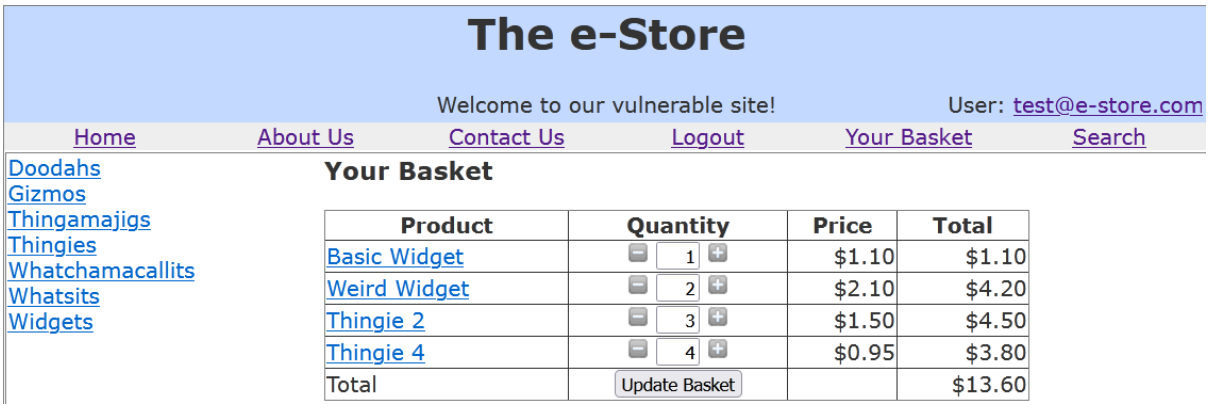


The traffic was forwarded after setting the b_id, it can be seen from the screenshot that access to the basket of the test@e-store.com account. This process can be repeated for all other accounts except the admin account which returned an error message. From this score page this challenge is completed and has turned green.

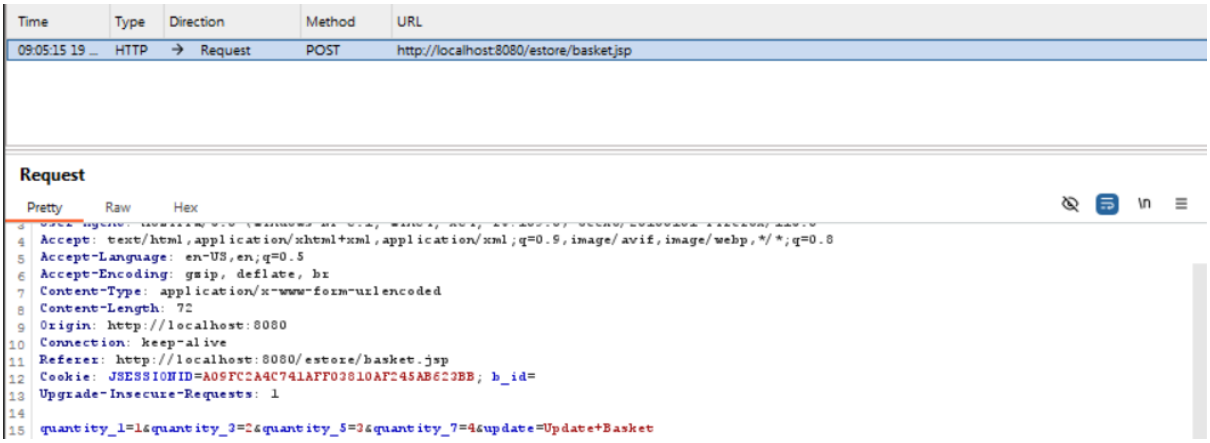


1.6 Challenge 9: Get the store to owe you money

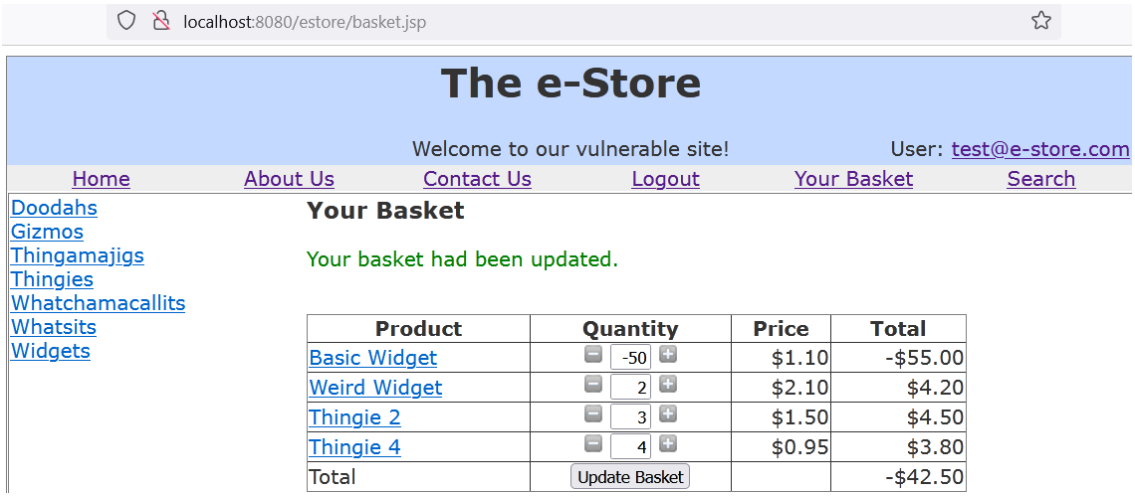
In this challenge the test@e-store.com account was used to get the store to owe you money, to achieve this Burp will be used for traffic capturing. The first thing that was done after logging in with the test@e-store.com email, ‘’Your Basket’’ link was clicked and it can be seen that by default the account has a balance of $13.60 as seen on the screenshot.



To get the store to owe you, the quantity of the first product (Basic Widget) was changed to 50, to do this the traffic was intercepted from Burp and it can be seen in line 15 that quantity_1=1 from the screenshot, we change that to quantity_1=-50 and forward the traffic.

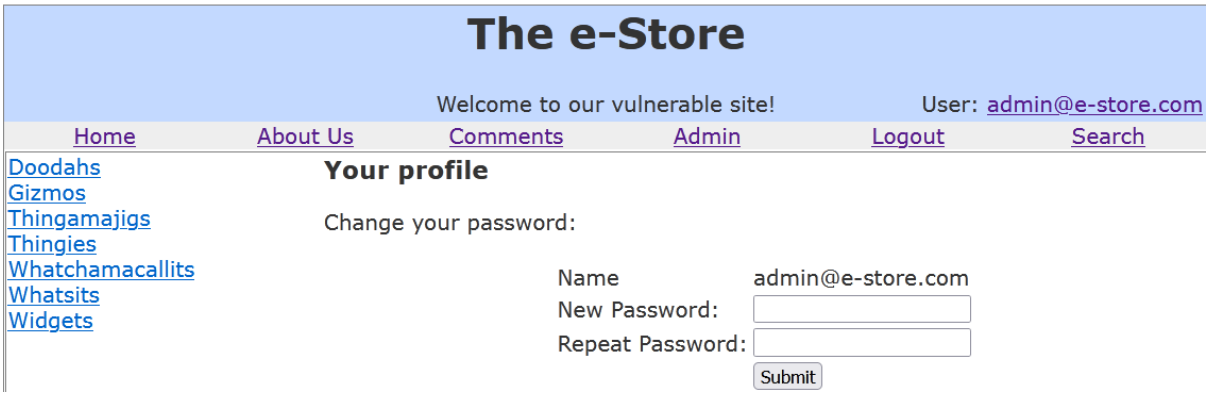


And sure enough it can be seen from the screenshot below, the store is now owing $42.50. From the score board this shows that the task has been completed.

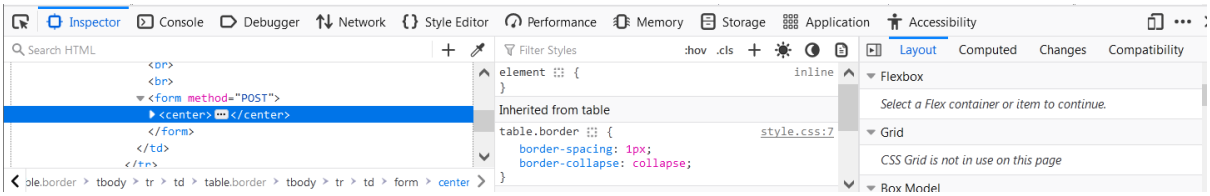


1.7 Challenge 10: Change your password via a GET request

In this challenge, the goal is to change the password for the account associated with ‘’admin@e-store.com’’. Access to the web application had already been gained without a password by using an SQL injection technique from challenge 1. From the top right corner of the page, the user email is clicked, leading to the password change page, as shown in the screenshot below.



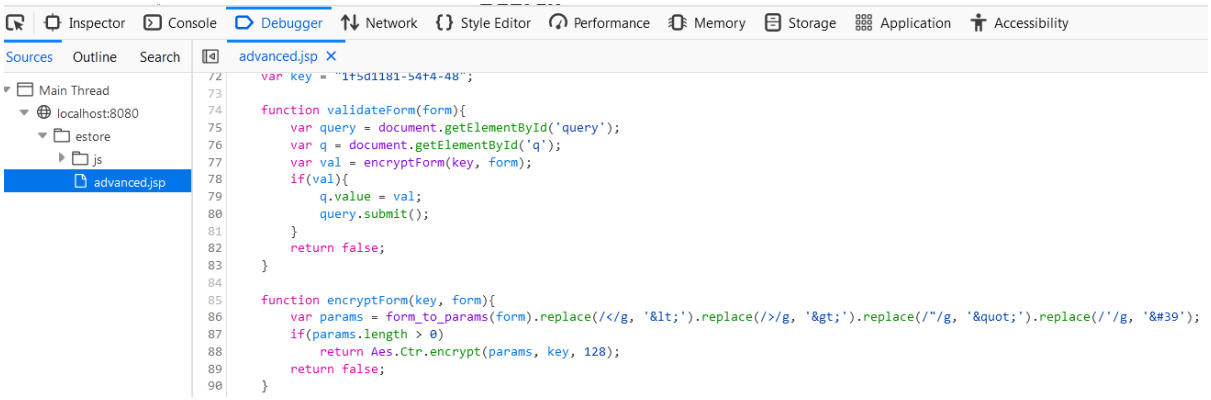
The page source was viewed, and it reveals that the password change form uses a POST request, which is normally used to send data to the server. To complete the challenge, this POST request is modified to a GET request. This is done by inspecting the password change page. Unlike the view source option, the inspect page tool allows direct editing. The request method is changed from POST to GET, and the password is updated to a chosen value, in this case, ‘’hack247’’ was used.



Upon clicking on the submit link, the password is successfully changed. This process can be repeated for any of the e-store accounts. From the scoreboard, it can be confirmed that the task is completed successfully, as the challenge indicator has turned green.

1.8 Challenge 11: Conquer AES encryption, and display a popup using: <script>alert("H@cked A3S")</script>

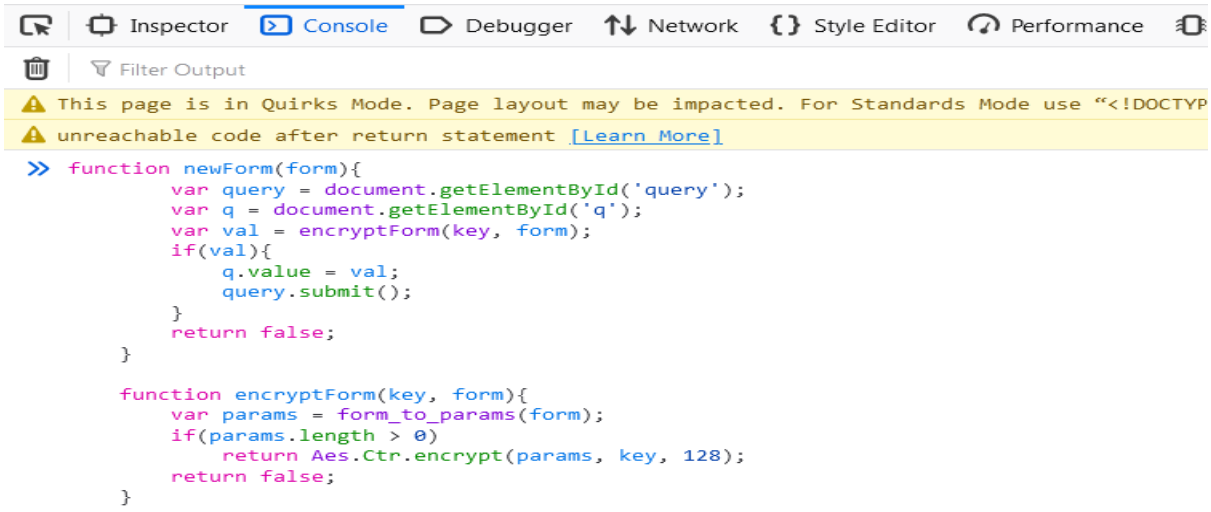
The goal of this challenge is to inject the malicious code ‘’<script>alert("H@cked A3S")</script>’’ into the e-store web application and trigger its execution, despite the use of AES encryption. To accomplish this, the search page of the e-store web application is accessed, followed by navigation to the advanced search section. The page is then inspected, and the debugger tab is opened. The JavaScript code that is responsible for the page is identified, this code collects form data, sanitizes it to prevent XSS attacks, and then subsequently encrypts it using AES. The code can be seen in the screenshot below.



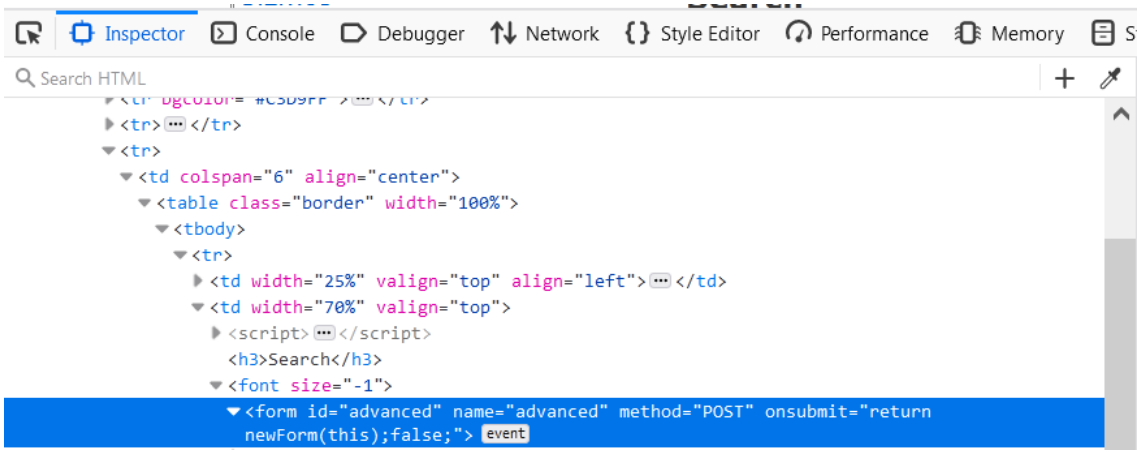
To bypass the sanitization, the JavaScript code is copied and pasted into the console tab, followed by two important modifications:

1. The function name ‘’validateForm’’ is changed to ‘’newForm’’ to prevent conflicts with the original function, as shown in the screenshots below.
2. The sanitization logic is removed, allowing inputs such as ‘’<script>’’ to bypass filtering and be submitted without modification.

These changes can be seen in the screenshots provided below.



Furthermore, from the inspector tab, the function was updated to use the recently modified version. Specifically, the ‘’validateForm’’ function was replaced with ‘’newForm’’, as shown in the screenshot below.



The payload command ‘’<script>alert("H@cked A3S")</script>’’ is entered into the ‘Type’ input field, which was followed by clicking the search button. This raw input bypassed the sanitization and is then encrypted. Furthermore, on the server side, the input is decrypted and inserted back into the webpage without proper validation, resulting in the execution of the injected JavaScript code. This resulted to a popup displaying "H@cked A3S", and the challenge scoreboard turns green, indicating successful completion of the challenge.

1.9 Challenge 12: Conquer AES encryption and append a list of table names to the normal results

In this challenge, the similar steps from Challenge 11 are followed; however, instead of injecting malicious JavaScript, an SQL injection is performed. The payload ***‘’xxxx' union select (select limit 3 1 table_name from information_schema.system_tables order by table_name),2,3,4,5 from products – 123”*** is entered into the ‘’Type’’ input field. This results in the display of the table names, as demonstrated in the video.

The injection is successful due to the modifications made to the JavaScript, specifically the removal of input sanitization. As a result, the server decrypts and processes the input without validation. This behaviour demonstrates a critical vulnerability where the server blindly trusts decrypted input, which enables both Cross-Site Scripting (XSS) and SQL injection attacks.

Link to video demonstration of challenge 12 : Meeting with USI EKEBOR (2332666)-20250322_021110-Meeting Recording.mp4

2.0 Task 2: Defensive Security

2.1 Tomcat Configuration

2.1.0 Disable Directory Listing

Disabling directory browsing is a security practice that prevents unauthorized access to files by stopping accidental exposure of sensitive data. It reduces risks such as directory traversal attacks and path manipulation. Using a servlet for role management, authentication, and session control ensures no error messages or information is exposed by maintaining a blank page. The directory listing was disabled by access the web.xml file in C:\xampp\tomcat\webapps\estore\WEB-INF\web.xml, this file was modified with the code in the screenshot below

```
<!-- Disable Directory Listings -->
    <servlet>
        <servlet-name>default</servlet-name>
        <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
    </servlet>
```

2.1.1 Restrict Access to Sensitive Data

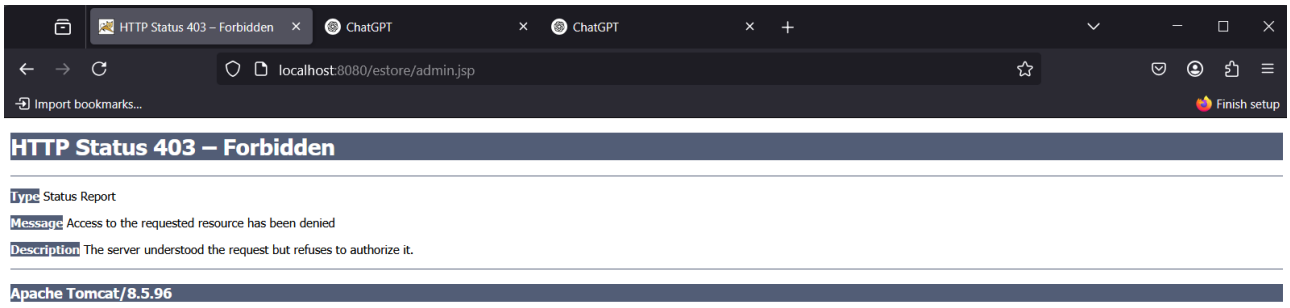
Disabling directory listing and restricting access to specific pages ensures that only authorized personnel can access critical features. This includes limiting access by IP, requiring authentication, and restricting unauthorized users, making sensitive information unavailable. These security measure would primarily protect e-stores from unauthorized access and potential breaches.

To do this the web.xml file is accessed from C:\xampp\tomcat\webapps\estore\WEB-INF\web.xml, the code is modified. The screenshot below shows the code that restrict access to the admin page data.

```
<!-- Restrict Access to admin.jsp -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>AdminPage</web-resource-name>
            <url-pattern>/admin.jsp</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
```

Consequently, when access to hidden data is requested as done in challenge 4, instead of showing the result that displayed earlier, an error was displayed as shown in the screenshot below.



2.1.2 Tomcat Secure Connections

To achieve this, two important configurations were implemented

2.1.2.0 HTTPS Enforcement

To achieve this, a security constraint that enforces HTTPS across the entire application using "<transport-guarantee>CONFIDENTIAL</transport-guarantee>" is added to the web.xml file located at C:\xampp\tomcat\webapps\estore\WEB-INF. as seen below.

```
<!-- ===== 4. NEW HTTPS ENFORCEMENT ===== -->
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>EntireApplication</web-resource-name>
            <url-pattern>/*</url-pattern> <!-- Applies to ALL URLs -->
        </web-resource-collection>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
```

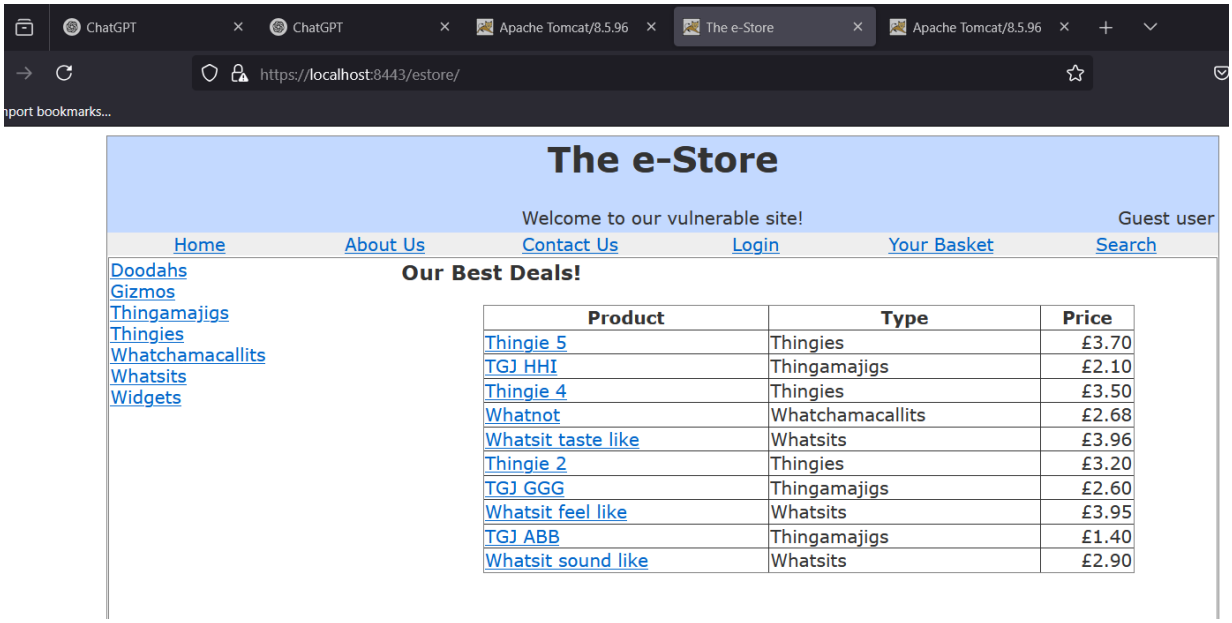
This added code will ensure that all client-server communication is encrypted which protects data from being intercepted during transmission. The code was not originally in the e-store which allows both HTTP and HTTPS access, but the HTTPS enforcement improves security of the e-store which aligns with the best practices for web applications development.

2.1.2.1 Configuring HTTPS and SSL/TLS

To configure HTTPS for Apache Tomcat, a self-signed SSL certificate must be generated and referenced in the Tomcat configuration. The first step is to access the Java keytool, which is in *C:\Program Files\Java\jdk-11\bin* from cmd. Then *keytool -genkey -keyalg RSA -alias tomcat -keystore C:\xampp\tomcat\conf\keystore.jks -storepass estore123 -validity 3650* was ran to generate a keystore.jks file which was saved in *C:\xampp\tomcat\conf*, this keystore contains the self-signed certificate. During certificate generation, details like organization and country code were entered. Furthermore, the Tomcat configuration file located at C:\xampp\tomcat\conf\server.xml was edited to include an SSL connector as shown in the screenshot below.

```
    -->
<Connector
  port="8443"
  protocol="org.apache.coyote.http11.Http11NioProtocol"
  maxThreads="150"
  SSLEnabled="true"
  scheme="https"
  secure="true"
  keystoreFile="C:\xampp\tomcat\conf\keystore.jks"
  keystorePass="estore123"
  clientAuth="false"
  sslProtocol="TLS"
/>
```

As seen above the connector is configured with the full path to the keystore file, along with the password set during creation. Once this is done and saved, Tomcat is restarted, and the HTTPS access can be verified at *https://localhost:8443/estore/* as seen below.



2.2 Source Code Vulnerabilities

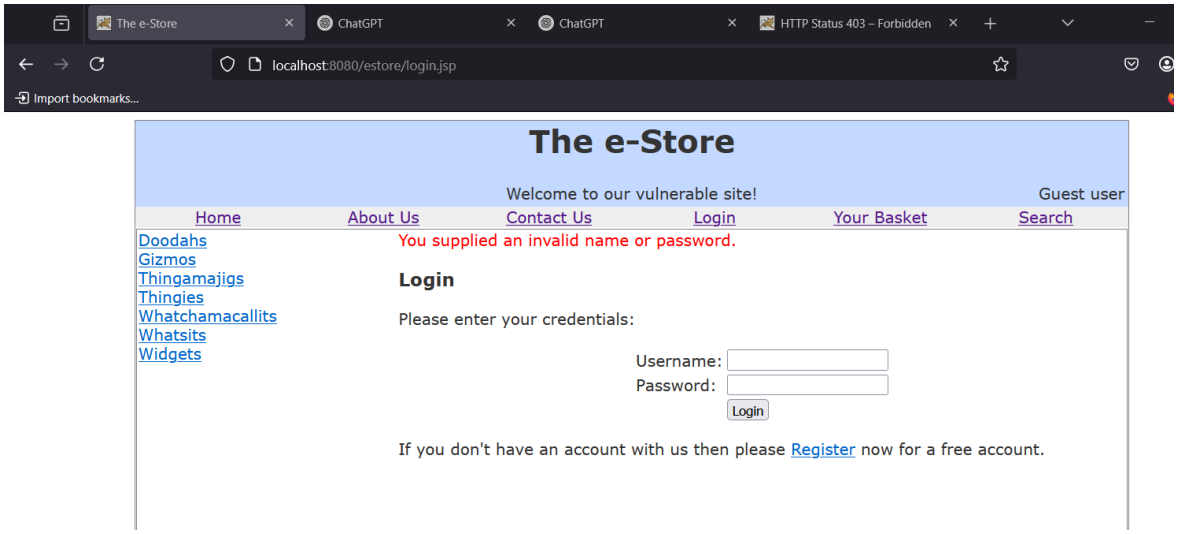
2.2.0 SQL Injection Mitigation

Prepared statements are a critical defence against SQL injection attacks which ensures secure database queries. Developers define the SQL structure first, then bind user inputs as parameters, preventing malicious input from being executed as code. This approach would enhance readability and prevents altering the original intent of the query.

The login.jsp page was vulnerable to SQL injection due to the dynamically constructed SQL queries. To fix this, the code at C:\xampp\tomcat\webapps\estore\login.jsp was modified to use prepared statements, ensuring user inputs are treated as literals. This can be seen from the screenshot below

```
36
37  if (request.getMethod().equals("POST") && username != null && password != null) {
38      PreparedStatement pstmt = null; // *Using PreparedStatement instead of Statement*
39      ResultSet rs = null;
40
41      try {
42          // *Fix: Use PreparedStatement to prevent SQL Injection*
43          String query = "SELECT * FROM Users WHERE name = ? AND password = ?";
44          pstmt = conn.prepareStatement(query);
45          pstmt.setString(1, username);
46          pstmt.setString(2, password);
47
48          rs = pstmt.executeQuery();
49
50          if (rs.next()) {
51              loggedIn = true;
52              debug = "Logged in";
53
54              // Store credentials in session
55              String userid = String.valueOf(rs.getInt("userid"));
56              session.setAttribute("username", rs.getString("name"));
57              session.setAttribute("userid", userid);
58              session.setAttribute("usertype", rs.getString("type"));
59
60              // *Fix: Securely update the scores*
61              String task = "";
62              if ("3".equals(userid)) {
63                  task = "LOGIN_TEST";
64              } else if ("1".equals(userid)) {
65                  task = "LOGIN_USER1";
66              } else if ("2".equals(userid)) {
67                  task = "LOGIN_ADMIN";
```

As a result of this, when a hacker inputs "test@e-store.com' OR '1'='1", the SQL engine treats it as text, preventing unauthorized query execution, the response can be seen below in the screenshot.

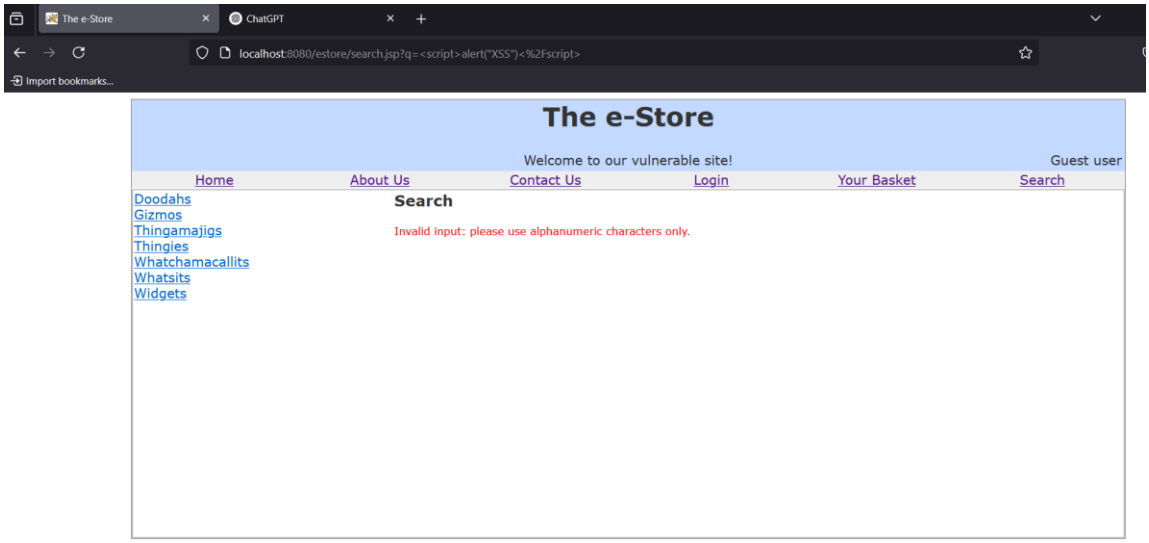


2.2.1 XSS Mitigation

To mitigate the XSS vulnerability in the e-store, the input validation and output encoding was applied to the search.jsp, register.jsp and advanced.jsp files located in C:\xampp\tomcat\webapps\estore. Similar measures were taken to achieve this; however, emphasis will be laid on the search.jsp file for proper illustration. First, the search.jsp file was opened and the file user input functionality was modified in the code to allow and validate only alphanumeric characters and basic punctuation, this will restrict any potentially dangerous input, this can be seen from the screenshot below.

```
31
32  <%
33  String query = request.getParameter("q");
34  boolean isValid = true;
35  String errorMessage = "";
36
37  if (request.getMethod().equalsIgnoreCase("GET") && query != null) {
38
39      // Input validation (allow only letters, digits, space, basic punctuation)
40      if (!query.matches("^[a-zA-Z0-9\\s,.'\"!?()-]{1,100}$")) {
41          isValid = false;
42          errorMessage = "Invalid input: please use alphanumeric characters only.";
43      }
44
45      if (isValid) {
46          // Optional: detect specific XSS challenge payloads for score logging
47          if (query.replaceAll("\\s", "").toLowerCase().contains("<script>alert(\"xss\")</script>")) {
48              conn.createStatement().execute("UPDATE Score SET status = 1 WHERE task = 'SIMPLE_XSS'");
49          }
50
51          // Output safe version of query
52          String safeQuery = StringEscapeUtils.escapeHtml4(query);
53  %>
```

Secondly, changes were made in the code to ensures that user inputs are HTML encoded, this confirms that user search input are safely displayed by changing any special character into a safe version, so it is shown as a text instead of the special character being executed as a code. Furthermore, the code is also modified to escape any potentially dangerous character before they are displayed on the web page from the database. When the payload used to complete challenge 6 was applied after this modification, displaying XSS popup on the search page became unsuccessful and returned invalid input as seen on the screenshot below.



These modifications made to the highlighted .jsp files prevent any harmful code from being executed by making sure the user input and data are displayed as safe text on the e-store webapp.

3.0 Task 3: Legal, Ethical, Social and Professional (LESP) Issues

Deploying the e-store webapp which is vulnerable to issues like SQL injection, cross-site scripting (XSS), and the bypassing of AES encryption could result to significant legal, ethical, social, and professional (LESP) challenges. To mitigate these risks and ensure the safety and trustworthiness of its infrastructure and users, it is essential to adhere to the legal requirements, act ethically, be socially responsible and follow professional standards. Here is an analysis of LESP for the e-store.

3.1 Legal Requirements

Compliance with Data Protection Laws

The e-store processes personal data such as names, email addresses, and purchase details. UK GDPR and the Data Protection Act 2018 require organisations to protect personal data through appropriate technical and organisational measures. Security flaws such as SQL injection and XSS expose stored data to unauthorised access. In 2018, British Airways suffered a data breach affecting over 400,000 customers and received a significant fine from the Information Commissioner’s Office due to poor security controls. This case demonstrates the legal consequences of failing to secure customer data.

Payment Security and PCI DSS

If a company processes card payments which the e-store does, it must comply with the Payment Card Industry Data Security Standard. PCI DSS requires secure storage, processing, and transmission of cardholder data. Weak encryption or input validation failures place payment data at risk. In 2013, Target experienced a major payment card breach caused by weak internal controls, leading to financial losses and regulatory scrutiny. Failure to meet PCI DSS obligations exposes the organisation to penalties and loss of payment processing privileges.

Computer Misuse Act 1990

Operating an insecure system increases the likelihood of unauthorised access offences under the Computer Misuse Act. While attackers commit the offence, organisations still face scrutiny if poor security enabled the attack. Preventable vulnerabilities weaken the organisation's legal position during investigations.

Copyright, Designs and Patents Act 1988

The e-store relies on third party libraries, frameworks, and media assets. Improper use of licensed software or images breaches copyright law. In 2017, Getty Images pursued legal action against several companies for unauthorised image use, resulting in settlements and reputational harm. Proper licence management and attribution remain legal obligations.

3.2 Ethical Considerations

Duty to Protect Users

The e-store company holds a clear ethical responsibility to protect users from harm. Leaving known vulnerabilities unresolved exposes users to fraud, identity theft, and financial loss. The ACM and IEEE Codes of Ethics emphasise protection of the public and avoidance of harm as core professional duties. Ignoring known weaknesses violates these principles.

Transparency and Responsible Disclosure

Ethical practice requires prompt disclosure of security incidents. Informing affected users supports trust and enables protective action. Uber faced criticism in 2017 after concealing a breach affecting millions of users, leading to regulatory penalties and public backlash. Concealment undermines ethical responsibility and public confidence.

User Consent and Data Choice

Users must control how their data is used. Ethical handling of opt in and opt out choices for marketing and data sharing supports user autonomy. Clear consent mechanisms align with ethical expectations and legal obligations.

3.3 Social Responsibility

Protecting Public Trust

Barauskaite and Streimikiene (2021) highlight the role of businesses in supporting societal wellbeing beyond profit. An insecure e-store damages trust in online commerce. Large scale breaches such as the Equifax incident in 2017 reduced public confidence in digital services across multiple sectors. Securing the platform supports trust in digital transactions and protects the wider online ecosystem.

Accessibility and Inclusion

Social responsibility includes accessibility for all users. Compliance with Web Content Accessibility Guidelines supports users with disabilities. In 2019, Domino's Pizza faced legal action in the United States due to an inaccessible website. While the case occurred outside the UK, it highlights growing expectations for accessible digital services. An accessible e-store improves inclusion and reduces legal and reputational risk.

3.4 Professional Standards

Following Industry Best Practices

There are multiple professional bodies like the OWASP (Open Web Application Security Project) that provide the best practice guideline for securing web applications. By building the e-store with the OWASP recommended practices in mind, would eliminate any of the security issues identified in task 1 and ensures that the system is designed to mitigate known risks like SQL injection and XSS identified in the completed challenges. These best practices are not just a recommendation but an expectation that the e-store webapp must meets the minimum-security standards before deployment for use.

Risk Management and Vulnerability Remediation

The e-store webapp could implement professional standards which would address any security issues, this could be measures like continuous monitoring the system and early deployment of security patches for any identified vulnerabilities. Furthermore, cybersecurity professionals should be engaged with the primary aim of conducting regular security audits, vulnerability scans, and applying any important patches. Regular vulnerability scanning, patch management, and security testing reduce exposure over time. Frameworks such as ISO 27001 and NIST provide structured approaches for managing information security risks. Organisations such as Maersk strengthened their security posture after the NotPetya attack by adopting stronger governance and monitoring controls. Applying these frameworks before deployment supports resilience and professional accountability.

References

ACM Code of Ethics and Professional Conduct.

BARAUSKAITE, G. and STREIMIKIENE, D., 2021. Corporate social responsibility and financial performance of companies: The puzzle of concepts, definitions and assessment methods. Corporate Social Responsibility and Environmental Management, 28(1), pp. 278–287.

CHAT GPT, . Chat GPT. [online]. Available from: https://chatgpt.com/

CHAT GPT, . Tomcat Source Code Modification. [online]. Available from: https://chatgpt.com/share/67e9d283-255c-800c-82f8-54306040f638

DEEPSEEK, . Deepseek. [online]. Available from: https://chat.deepseek.com/

FRIEDMAN, M., 2007. The social responsibility of business is to increase its profits. In: Anonymous Corporate ethics and corporate governance. Springer. pp. 173–178.

General Data Protection Regulation (GDPR).

IEEE Code of Ethics.

OWASP Secure Coding Practices.

STOBIERSKI, T., 2021. What Is Corporate Social Responsibility? [online]. Available from: https://online.hbs.edu/blog/post/types-of-corporate-social-responsibility

Data Protection Act 2018.