ASSIGNMENT-2

NAME-KHUSHBOO CHAUDHARY

ROLL NO.-U24CS038

AIM: To study and demonstrate Cross-Site Scripting (XSS) attacks in web applications by analyzing a given case study, identifying vulnerable input handling, and observing the behavior of reflected and stored XSS attacks.

Case Study

1**: Vulnerable User Feedback Web Page**

• Scenario:

– A web application allows users to submit textual feedback using an input form.

– The submitted feedback is dynamically displayed on a web page using HTML and JavaScript.

– The application assumes that all user input is safe and does not apply any validation or sanitization.

– The feedback submitted by one user is visible to other users accessing the page.

• It is observed that certain inputs cause unexpected browser behavior, such as dialog boxes appearing automatically when the page is accessed or refreshed.

Vulnerable Web Page (Demo)

Demonstration

Input this into the feedback box:

<script>alert("The site is under XSS attack")</script>

Observed Behavior

- **A browser dialog box appears.**
- **The script runs because the input is directly inserted into innerHTML.**

2: Implementation of Cross-Site Scripting (XSS)

• Develop a simple web page that accepts user input and displays it dynamically.

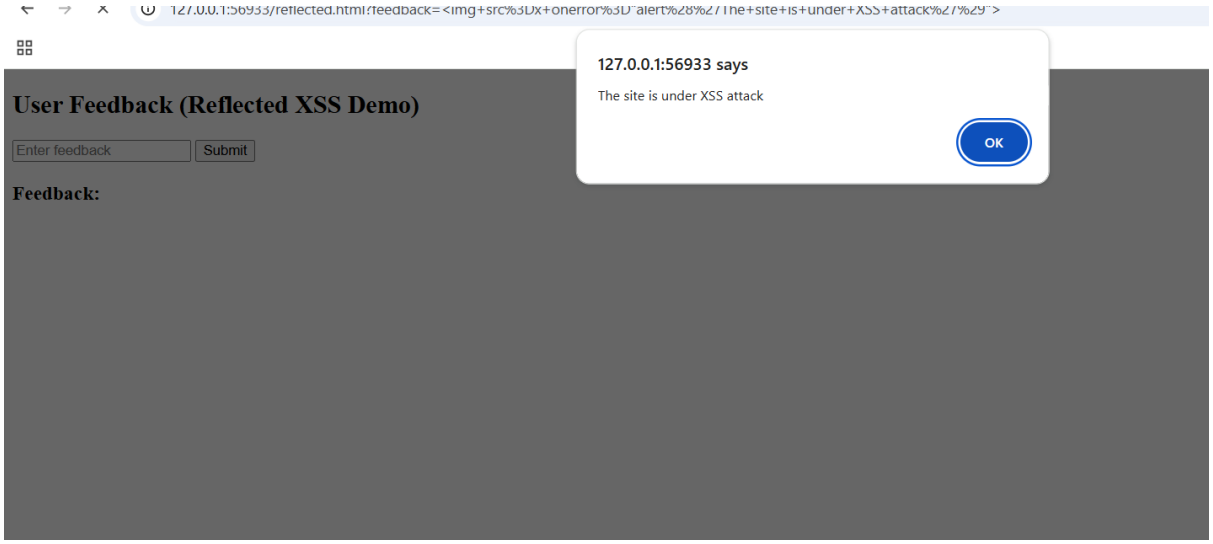
• Analyze how unsafe handling of user input results in execution of injected scripts.

• Demonstrate a scenario where a browser dialog box appears with the message:

"The site is under XSS attack"

• Identify the portion of the application logic responsible for the vulnerability.

```
<!DOCTYPE html>
<html>
<head>
  <title>User Feedback - Reflected XSS</title>
</head>
<body>
  <h2>User Feedback (Reflected XSS Demo)</h2>

  <form method="GET">
    <input type="text" name="feedback" placeholder="Enter feedback">
    <button type="submit">Submit</button>
  </form>

  <h3>Feedback:</h3>
  <div id="output"></div>

  <script>
    const params = new URLSearchParams(window.location.search);
    const feedback = params.get("feedback");

    if (feedback) {

      document.getElementById("output").innerHTML = feedback;
    }
  </script>
</body>
</html>
```



3: Reflected Cross-Site Scripting (XSS)

• Analyze a scenario where malicious input is immediately reflected in the server response.

• Demonstrate how the injected script executes only when the crafted input is provided.

• Observe the scope and timing of the attack and identify the affected users.

4: Stored Cross-Site Scripting (XSS)

• Analyze a scenario where malicious input is stored in the system.

• Demonstrate how the injected script executes whenever the affected page is accessed.

• Compare the persistence and impact of stored XSS with reflected XSS.

```html
<!DOCTYPE html>
<html>
<head>
  <title>User Feedback - Stored XSS</title>
</head>
<body>
  <h2>User Feedback (Stored XSS Demo)</h2>

  <input id="fb" placeholder="Enter feedback">
  <button onclick="saveFeedback()">Submit</button>

  <h3>All Feedback:</h3>
  <div id="allFeedback"></div>

  <script>
    function saveFeedback() {
      let fb = document.getElementById("fb").value;
      let all = localStorage.getItem("feedback") || "";
      localStorage.setItem("feedback", all + "<p>" + fb + "</p>");
      showFeedback();
    }

    function showFeedback() {

      document.getElementById("allFeedback").innerHTML =
        localStorage.getItem("feedback");
    }

    showFeedback();
  </script>
</body>
</html>
```

5: Student Observations

• Differences observed between reflected and stored XSS attacks.

• Impact of XSS on confidentiality, authentication, and integrity.

• Reasons why improper input handling leads to serious security vulnerabilities.

• Key lessons learned from this experiment.

Differences Observed

Reflected XSS	**Stored XSS**
Runs only when special link/input is used	Runs automatically for all users
Not saved	Saved in storage/database
Temporary	Persistent
Lower impact	Higher impact

Impact of XSS

- Cookie/session theft
- Fake login forms
- Website defacement
- Malware injection
- Redirection to malicious sites

Why improper input handling is dangerous

- Browser trusts scripts inside HTML

- Attackers inject executable code
- Affects users without their knowledge

Key Lessons Learned

- Never trust user input
- Avoid inner HTML with raw data
- Always validate and encode
- Stored XSS is more dangerous than reflected XSS
- XSS breaks Confidentiality, Integrity, Availability