Lahore Garrison University

DHA Phase 6 Lahore

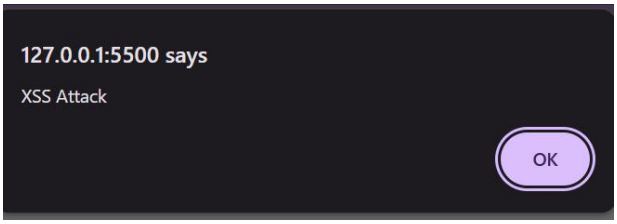
Student Name	**Abu Bakar**
Student Roll No	**Fa23/BSCS/272**
Section	**G**
Subject	**Web Technologies**
Instructor Name	**Mr. M Yousaf**

1. Demonstrating an XSS Payload on an Insecure Page

1.1 Vulnerable HTML Page:

```
<!DOCTYPE html>

<html>

<head>

   <title>XSS Demo - Insecure</title>

</head>

<body>

   <h2>Insecure Comment Box</h2>

   <form id="commentForm">

      <input type="text" id="commentInput" placeholder="Type something..." />

      <button type="submit">Post</button>

   </form>

   <div id="comments"></div>

   <script>

      document.getElementById('commentForm').addEventListener('submit',
function(e) {

         e.preventDefault();

         const input = document.getElementById('commentInput').value;

         document.getElementById('comments').innerHTML += "<p>" + input +
"</p>";

      });

   </script>

</body>

</html>
```

Output:



2. Implementing Input Sanitization With DOMPurify:

```
<!DOCTYPE html>
<html>
<head>
   <title>XSS Prevention - Secure</title>
   <!-- Include DOMPurify CDN -->
   <script
src="https://cdn.jsdelivr.net/npm/dompurify@3.0.0/dist/purify.min.js"></script>
</head>
<body>
   <h2>Secure Comment Box</h2>
   <form id="commentForm">
      <input type="text" id="commentInput" placeholder="Type something..." />
      <button type="submit">Post</button>
   </form>
   <div id="comments"></div>
   <script>
      document.getElementById('commentForm').addEventListener('submit',
function(e) {
         e.preventDefault();
         const input = document.getElementById('commentInput').value;
         const clean = DOMPurify.sanitize(input);
```

```
            const p = document.createElement('p');
        p.textContent = clean;
        document.getElementById('comments').appendChild(p);
    });
  </script>
</body>
</html>
```

Output:

3. CSRF Protection — Explanation & Token Mockup

3.1 What is CSRF?

Cross-Site Request Forgery (CSRF) forces a user to take unwanted actions on a site where they are authenticated (e.g., changing password, posting something).

Attackers exploit:

Browser auto-sending cookies (session cookies)

Malicious hidden forms or requests

3.2 CSRF Token Concept:

A CSRF token is a secret, random value generated by the server and included in forms.

The server verifies the token on form submission.

3.3 Frontend Example Including Token:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
 </head>
 <body>
    <form id="secureForm">
    <input type="hidden" id="csrfToken" value="mock-random-token-12345">
    <input type="text" id="comment" />
    <button>Post</button>
</form>
<script>
    document.getElementById("secureForm").addEventListener("submit", e => {
        e.preventDefault();
        const token = document.getElementById("csrfToken").value;
        console.log("Sending with CSRF token:", token);
        // Example simulated server check
        if (token !== "mock-random-token-12345") {
            alert("CSRF token invalid!");
        } else {
            alert("Form submitted securely!");
        }
    });
</script>
 </body>
 </html>
```

Output: