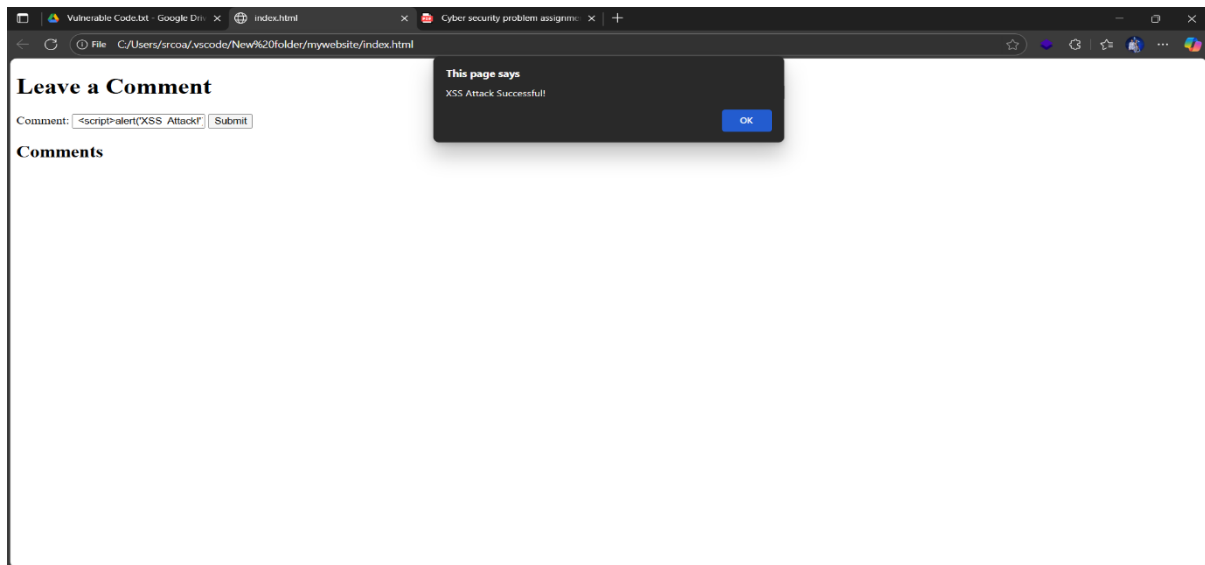
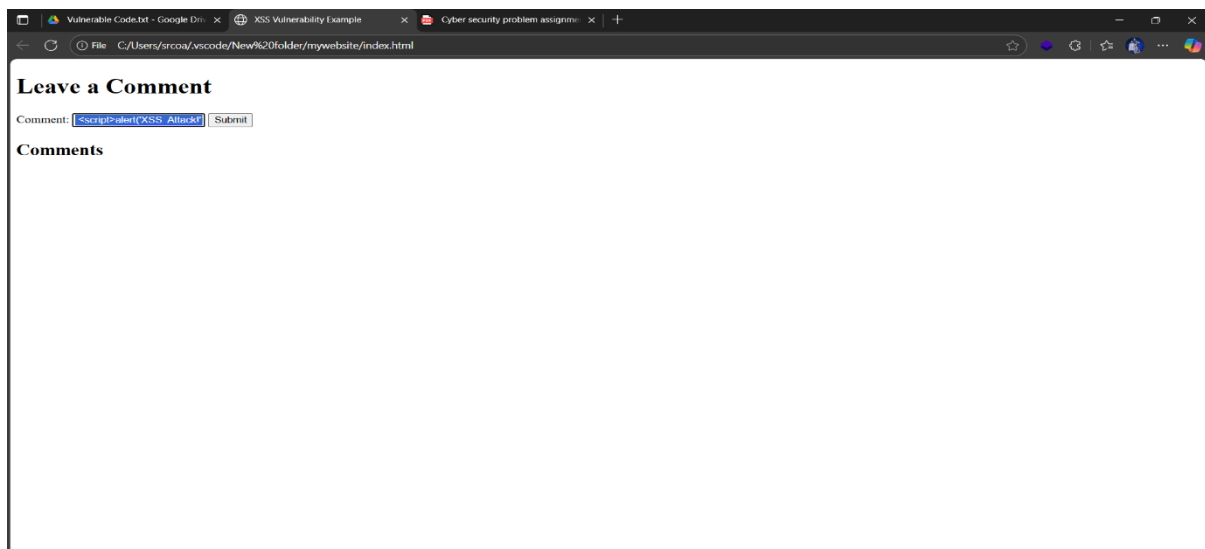


ASSIGNMENT:-2 Solve the XSS Vulnerability



Screenshot 1: show the Alert On Vulnerable programme



Screenshot 2 : input Vulnerable code in comment field

This code is vulnerable because Its show XSS Script Alert

**** Vulnerability Explanation ****

Problem Areas:

1. **innerHTML** usage:

- `var comment = document.getElementById('comment').value;`

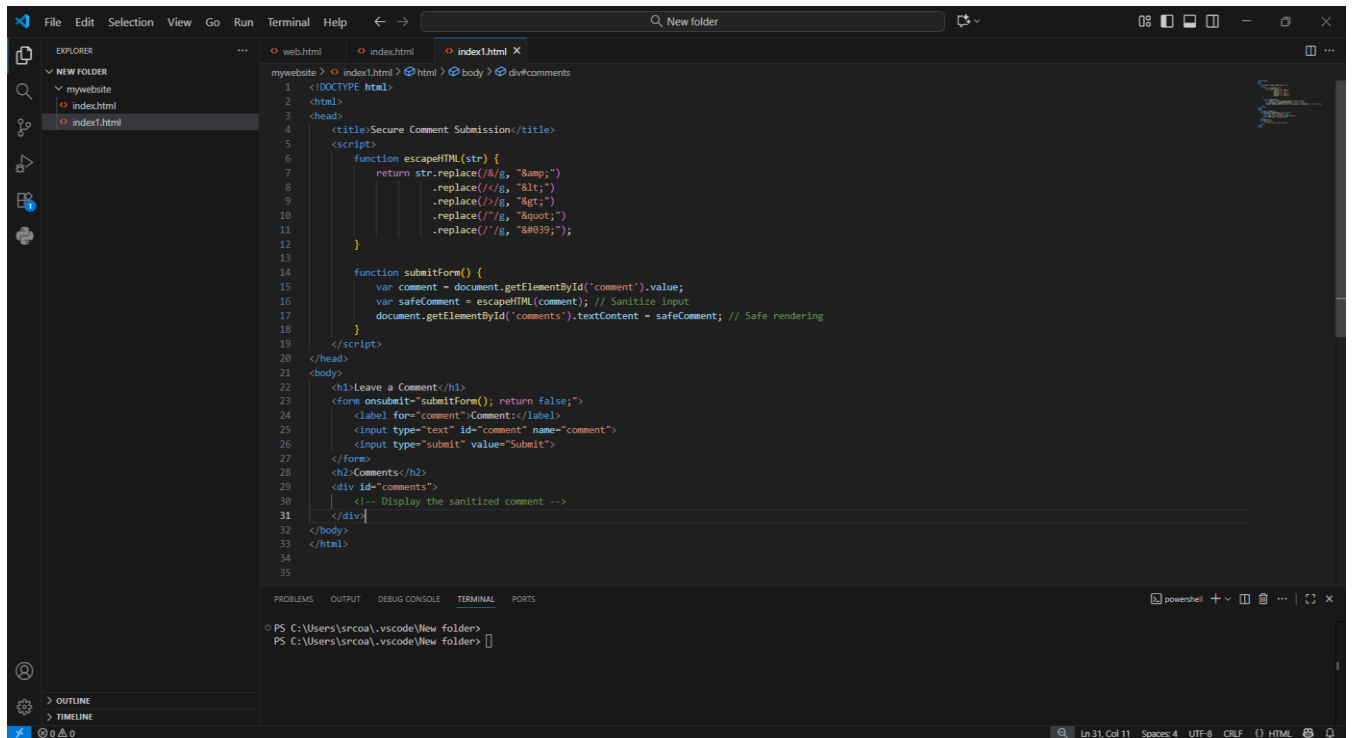
This directly injects user input into the DOM, allowing malicious scripts like `<script>alert('XSS');</script>` to execute.

2. **document.write();** with script injection:

- `document.write(scriptTag);`

This forcibly writes a script tag into the document, which is a classic XSS vector.

**** Secure Version (XSS Mitigation)**

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'mywebsite' with an 'index.html' file. The main editor area displays the content of 'index1.html'. The code is an HTML document with a title 'Secure Comment Submission' and a script block. The script defines an 'escapeHTML' function to sanitize user input by replacing special characters with their HTML entities. It then uses this function to sanitize a comment before displaying it. The HTML structure includes a form with a text input and a submit button, and a div to display the sanitized comment.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Secure Comment Submission</title>
5   <script>
6     function escapeHTML(str) {
7       return str.replace(/&/g, "&amp;")
8         .replace(/</g, "&lt;")
9         .replace(/>/g, "&gt;")
10        .replace(/"/g, "&quot;")
11        .replace(/'/g, "&#039;");
12    }
13
14    function submitForm() {
15      var comment = document.getElementById('comment').value;
16      var safeComment = escapeHTML(comment); // Sanitize input
17      document.getElementById('comments').textContent = safeComment; // Safe rendering
18    }
19  </script>
20 </head>
21 <body>
22   <h1>Leave a Comment</h1>
23   <form onsubmit="submitForm(); return false;">
24     <label for="comment">Comments:</label>
25     <input type="text" id="comment" name="comment">
26     <input type="submit" value="Submit">
27   </form>
28   <h2>Comments</h2>
29   <div id="comments">
30     <!-- Display the sanitized comment -->
31   </div>
32 </body>
33 </html>
```

Screenshot 3 : xss mitigation code on Vscode

Full Code :

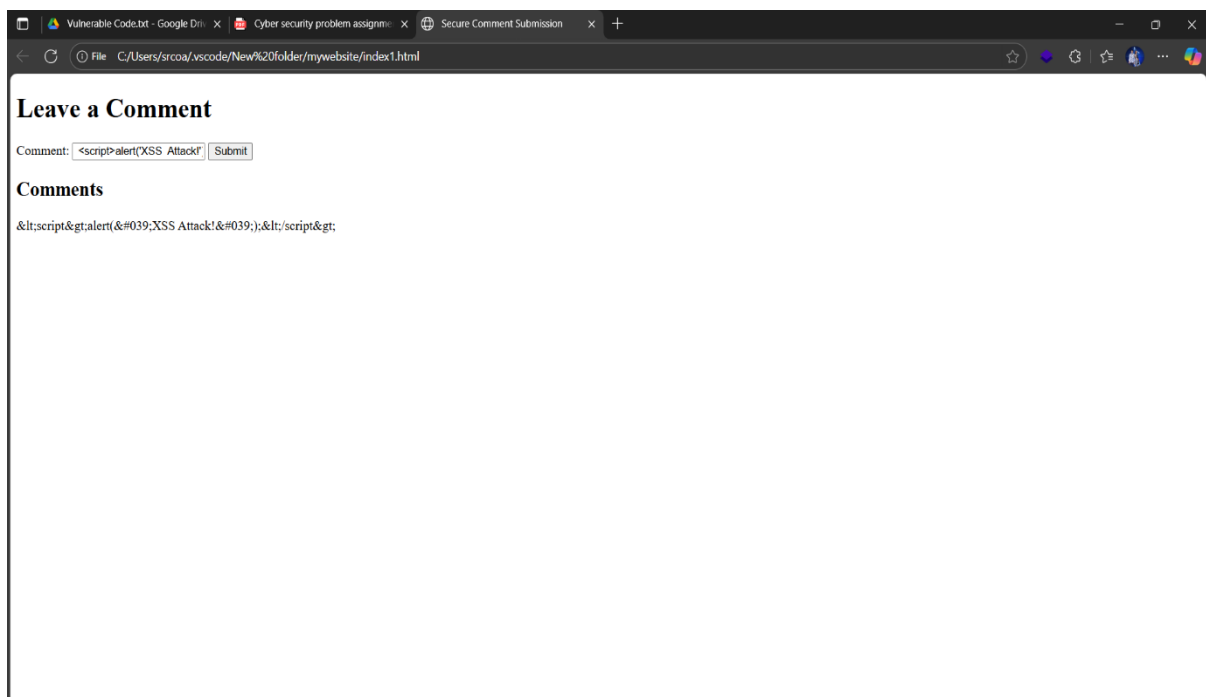
```
<!DOCTYPE html>
<html>
<head>
  <title>Secure Comment Submission</title>
  <script>
    function escapeHTML(str) {
      return str.replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");
    }
  </script>
```

```

        function submitForm() {
            var comment = document.getElementById('comment').value;
            var safeComment = escapeHTML(comment); // Sanitize input
            document.getElementById('comments').textContent = safeComment; //
Safe rendering
        }
    </script>
</head>
<body>
    <h1>Leave a Comment</h1>
    <form onsubmit="submitForm(); return false;">
        <label for="comment">Comment:</label>
        <input type="text" id="comment" name="comment">
        <input type="submit" value="Submit">
    </form>
    <h2>Comments</h2>
    <div id="comments">
        <!-- Display the sanitized comment -->
    </div>
</body>
</html>

```

Screenshot 4 : xss mitigation Code



Screenshot 5 : XSS Vulnerability Fixed Input Comment Field Sanitized

Alert not shown

@@ Why This Fix Works:

- Escaping HTML characters prevents script tags from being interpreted by the browser.
- Using `TextContent` instead of `innerHTML` ensures that even if malicious input slips through, it's treated as plain text—not executable code.
- Avoiding `document.write()` eliminates a major XSS vector