Cybersecurity Problem Assignment-2 Report

Internship Program: Cybersecurity

Submitted by: Vedant Sangamnere

Date: 16 / 08 / 2004

Introduction

This report outlines the identification and resolution of a Cross-Site Scripting (XSS) vulnerability in a given web application. XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users. The objective of this assignment is to identify the vulnerability and mitigate it by implementing a secure solution.

Problem Statement

The provided web page contains an XSS vulnerability due to improper handling of user input. Specifically, the user input is inserted into the HTML using the innerHTML method, which allows any HTML or JavaScript code to be executed in the user's browser. The goal is to fix this vulnerability to prevent malicious code execution.

Original Vulnerable Code

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
   <title>XSS Vulnerable Example</title>
</head>
<body>

   <h2>Enter Your Name:</h2>
   <form>
```

```
<input type="text" id="userInput" placeholder="Your Name">

<button type="button" onclick="displayInput()">Submit</button>
</form>

<p id="output"></p>

<script>
    function displayInput() {
        var input = document.getElementById("userInput").value;
        // Vulnerable to XSS
        document.getElementById("output").innerHTML = "Hello, " + input + "!";
    }
</script>

</body>
</html>
```

Vulnerability Analysis

The vulnerability in this code arises because the user input is directly embedded into the HTML content without any sanitization. This allows an attacker to inject malicious JavaScript code, which will be executed in the user's browser.

For example, input such as **<script>alert('XSS Attack!');</script>** will trigger a JavaScript alert pop-up when submitted, demonstrating the XSS vulnerability.

Solution

To mitigate this vulnerability, I implemented input sanitization and replaced the use of innerHTML with innerText. The sanitization function replaces special characters such as <, >, &, ", and ' with their corresponding HTML entities. This ensures that any user input is treated as plain text and cannot be executed as code.

Modified Secure Code

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Fixed XSS Example</title>
</head>
<body>

    <h2>Enter Your Name:</h2>
    <form>
        <input type="text" id="userInput" placeholder="Your Name">
        <button type="button" onclick="displayInput()">Submit</button>
    </form>

    <p id="output"></p>

    <script>
        function displayInput() {
            var input = document.getElementById("userInput").value;
            // Sanitize input to prevent XSS
            var sanitizedInput = sanitizeInput(input);
            document.getElementById("output").innerText = "Hello, " + sanitizedInput + "!";
        }

        // Sanitize the input by replacing dangerous characters
```

```
function sanitizeInput(input) {


 return input.replace(/&/g, "&amp;")
                    .replace(/</g, "&lt;")
                    .replace(/>/g, "&gt;")
                    .replace(/"/g, "&quot;")
                    .replace(/'/g, "&#039;");
      }
   </script>


</body>
</html>
```

Explanation of the Solution

The sanitizeInput() function replaces potentially dangerous characters with their corresponding HTML entities:

- < is replaced with <
- > is replaced with >
- & is replaced with &
- " is replaced with "
- ' is replaced with '

These replacements prevent any injected scripts from being executed. Furthermore, replacing innerHTML with innerText ensures that the content is displayed as plain text, not as HTML.

Testing the Solution

To verify that the vulnerability is resolved, I tested the fixed code using the following input:

<script>alert('XSS Attack!');</script>

After submitting this input, the page displayed the following:

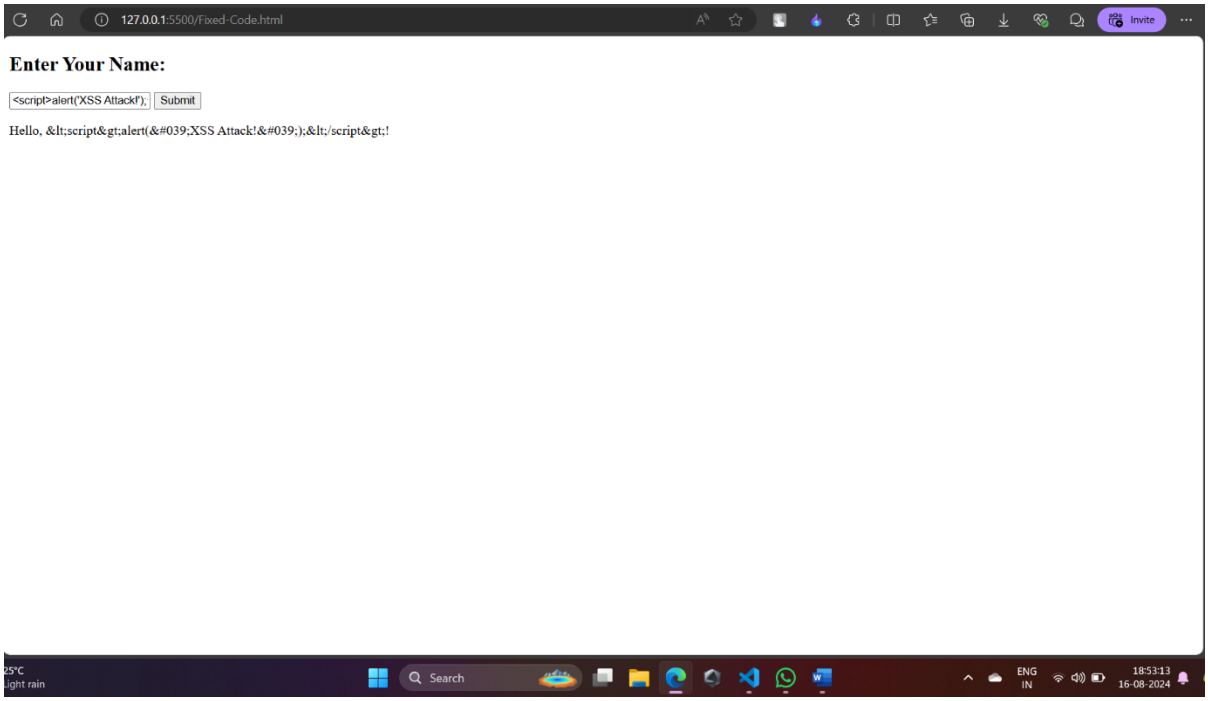
Hello, <script>alert('XSS Attack!');</script>

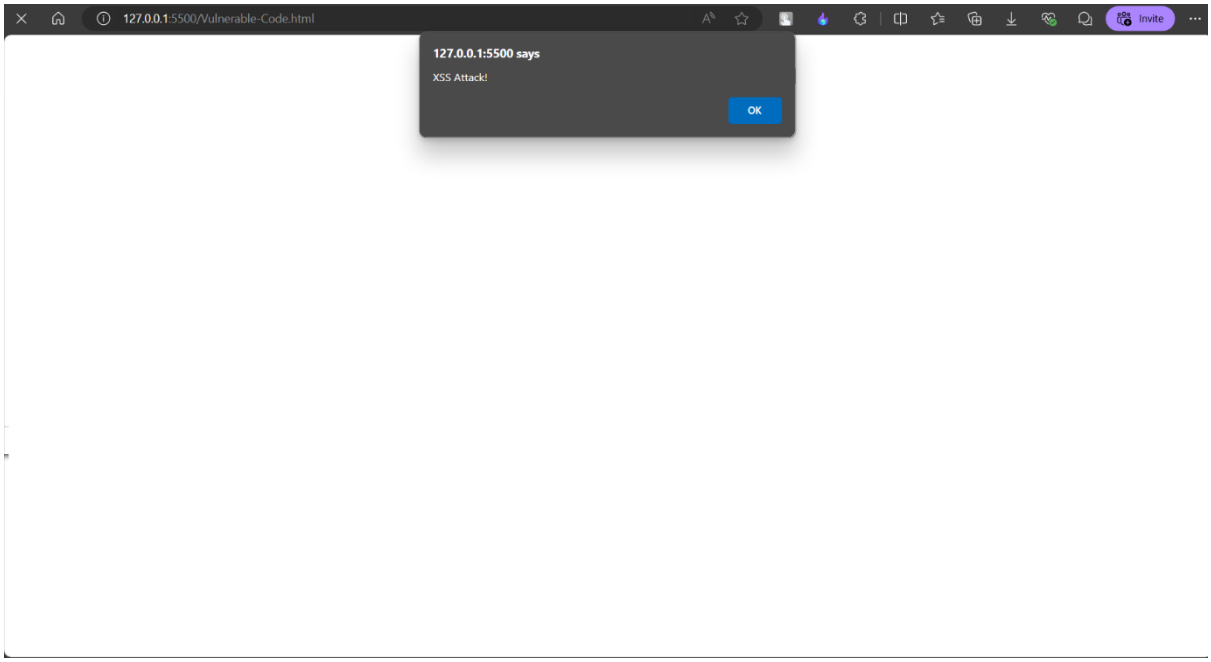
No JavaScript alert was triggered, confirming that the vulnerability was successfully mitigated.

Conclusion

By sanitizing user input and avoiding the direct insertion of HTML content, the XSS vulnerability was effectively mitigated. This solution prevents malicious scripts from executing in the user's browser, significantly improving the security of the web application.

Screenshots




127.0.0.1:5500/Vulnerable-Code.html
Invite
127.0.0.1:5500 says
XSS Attack!
OK