Lab 8 – Input Validation

Eshwar Kyatham

Northern Arizona University

ITC 505: Web Techonolgy

Troy L Adams

07/22/2024

Source Code

Website Link: <https://eshwarkyatham16.github.io/ITC505/lab-8/index.html>

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>Secure Form</title>

<link rel="stylesheet" href="styles.css" />

</head>

<body>

<h1>Secure Form</h1>

<form id="registrationForm" action="/submit" method="POST">

<label for="firstName">First Name \*</label>

<input type="text" id="firstName" name="firstName" />

<span class="error-message" id="firstNameError"></span>

<label for="lastName">Last Name \*</label>

<input type="text" id="lastName" name="lastName" />

<span class="error-message" id="lastNameError"></span>

<label for="email">Email Id\*</label>

<input type="email" id="email" name="email" />

<span class="error-message" id="emailError"></span>

<label for="password">Password \*</label>

<input type="password" id="password" name="password" />

<span class="error-message" id="passwordError"></span>

<label for="confirmPassword">Confirm Password \*</label>

<input type="password" id="confirmPassword" name="confirmPassword" />

<span class="error-message" id="confirmPasswordError"></span>

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

</form>

<script src="script.js"></script>

</body>

</html>

Styles.css

body {

*font-family*: Arial, sans-serif;

*margin*: 50px;

}

form {

*max-width*: 400px;

*margin*: 0 auto;

}

label {

*display*: block;

*margin-bottom*: 8px;

}

input[type="text"],

input[type="email"],

input[type="password"] {

*width*: 100%;

*padding*: 8px;

*margin-bottom*: 20px;

*border*: 1px solid #ccc;

*border-radius*: 4px;

}

input.error {

*border-color*: red;

}

.error-message {

*color*: red;

*margin-top*: -15px;

*margin-bottom*: 15px;

*display*: block;

}

button {

*padding*: 10px 20px;

*background-color*: #007bff;

*color*: white;

*border*: none;

*border-radius*: 4px;

*cursor*: pointer;

}

button:disabled {

*background-color*: #ccc;

*cursor*: not-allowed;

}

Scripts.js (With JavaScript validation, and XSS/SQLi prevention)

document

.getElementById("registrationForm")

.addEventListener("submit", *function* (*event*) {

*event*.preventDefault();

*const* firstName = document.getElementById("firstName").value.trim();

*const* lastName = document.getElementById("lastName").value.trim();

*const* email = document.getElementById("email").value.trim();

*const* password = document.getElementById("password").value.trim();

*const* confirmPassword = document

.getElementById("confirmPassword")

.value.trim();

*let* isValid = true;

document.querySelectorAll(".error-message").forEach(*function* (*el*) {

*el*.textContent = "";

});

document.querySelectorAll("input").forEach(*function* (*el*) {

*el*.classList.remove("error");

});

if (!firstName) {

isValid = false;

document.getElementById("firstNameError").textContent =

"First name is required.";

document.getElementById("firstName").classList.add("error");

}

if (!lastName) {

isValid = false;

document.getElementById("lastNameError").textContent =

"Last name is required.";

document.getElementById("lastName").classList.add("error");

}

if (!email) {

isValid = false;

document.getElementById("emailError").textContent =

"Email Id is required.";

document.getElementById("email").classList.add("error");

} else {

// Email validation

*const* emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailRegex.test(email)) {

isValid = false;

document.getElementById("emailError").textContent =

"Please enter a valid email address.";

document.getElementById("email").classList.add("error");

}

}

if (!password) {

isValid = false;

document.getElementById("passwordError").textContent =

"Password is required.";

document.getElementById("password").classList.add("error");

}

if (!confirmPassword) {

isValid = false;

document.getElementById("confirmPasswordError").textContent =

"Confirm password is required.";

document.getElementById("confirmPassword").classList.add("error");

} else if (password !== confirmPassword) {

isValid = false;

document.getElementById("confirmPasswordError").textContent =

"Passwords do not match.";

document.getElementById("confirmPassword").classList.add("error");

}

if (isValid) {

alert("Form submitted successfully!");

}

});

**XSS and SQLi**

Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a prevalent security vulnerability in web applications that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can execute arbitrary actions within the context of the user's session, potentially leading to a range of malicious outcomes.

Types of XSS Attacks

*Stored XSS:*

* Description: In Stored XSS attacks, the malicious script is permanently stored on the server. This could be in a database, message forum, or other persistent storage. When other users view the affected page, the malicious script is served as part of the page content.
* Example: An attacker posts a comment containing malicious JavaScript on a blog. When other users visit the blog, the script executes in their browsers, potentially stealing cookies or session tokens.

*Reflected XSS:*

* Description: Reflected XSS occurs when user input is immediately reflected off a web application without proper sanitization. This typically happens via URL parameters or form submissions. The malicious script is then executed in the victim’s browser.
* Example: An attacker crafts a URL with a malicious payload in a query parameter. When a user clicks this link, the script in the URL executes because it is reflected in the web page’s response.

*DOM-based XSS:*

* Description: DOM-based XSS exploits vulnerabilities in client-side code (JavaScript) rather than server-side code. The attack involves modifying the Document Object Model (DOM) of the page to execute malicious scripts.
* Example: An attacker can manipulate a URL fragment (e.g., #hash) to inject a script that alters the DOM in the user's browser, causing unintended actions like data exfiltration.

Mitigation Measures

*Input Validation:*

* Explanation: Ensure that all user inputs are validated and sanitized before processing. This involves checking input data for correctness and removing any potentially harmful characters or scripts.
* Implementation: Use libraries or frameworks that automatically handle input sanitization. For example, in JavaScript, you can use functions like encodeURIComponent to escape user inputs.

*Output Encoding:*

* Explanation: Encode output data before rendering it in the browser. This ensures that any data that could be interpreted as executable code is instead treated as plain text.
* Implementation: Utilize functions like htmlspecialchars in PHP or similar methods in other languages to escape special characters such as <, >, &, and ", which could be used in XSS attacks.

*Content Security Policy (CSP):*

* Explanation: Implement a CSP to restrict the types of content that can be loaded and executed by a web page. CSP can mitigate the impact of XSS attacks by preventing the execution of unauthorized scripts.
* Implementation: Set up a CSP header in your web server configuration, specifying allowed sources for scripts, styles, and other content.

SQL Injection (SQLi)

SQL Injection (SQLi) is a code injection attack that allows an attacker to interfere with the queries an application makes to its database. By injecting malicious SQL code, attackers can gain unauthorized access to or manipulate database data.

*Types of SQL Injection In-band SQLi:*

* Description: In this type of SQLi, the attacker uses the same communication channel to launch the attack and retrieve results. This is the most common form and can be performed using error-based or union-based techniques.
* Example: An attacker inputs a SQL payload into a login form to bypass authentication or extract data from the database.

*Inferential (Blind) SQLi:*

* Description: In Blind SQLi, the attacker cannot see the results of the query directly. Instead, they infer information based on the application's responses, such as changes in behavior or error messages.
* Example: An attacker might modify a query to check if certain conditions are true based on the application's response, inferring the structure and contents of the database.

*Out-of-band SQLi:*

* Description: Out-of-band SQLi involves using different channels for the attack and data retrieval. This method is less common but can be used when other types of SQLi are not possible.
* Example: An attacker could use SQL queries to send data to an external server controlled by the attacker, which then retrieves the data via HTTP requests.

Mitigation Measures

*Prepared Statements:*

* Explanation: Prepared statements use parameterized queries to separate SQL code from data input. This prevents attackers from injecting malicious SQL code into queries.
* Implementation: Use language-specific libraries or frameworks that support prepared statements, such as mysqli or PDO in PHP, or parameterized queries in SQL Server.

*Stored Procedures:*

* Explanation: Stored procedures encapsulate SQL queries in database routines, reducing the risk of SQL injection by separating data and code.
* Implementation: Define and call stored procedures in your database, and ensure that they use parameterized inputs.

*Input Validation:*

* Explanation: Validate and sanitize all user inputs before using them in SQL queries. Ensure that inputs conform to expected formats and types.
* Implementation: Use validation libraries or frameworks to check input data against predefined rules and sanitize inputs to remove potentially harmful content.

Conclusion

Understanding and mitigating XSS and SQL Injection vulnerabilities are crucial for developing secure web applications. By implementing proper input validation, output encoding, and using prepared statements or stored procedures, developers can protect their applications from these common and dangerous attacks.

**TryHackMe SQLi**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Citation

Kyle, S. (2023, Jan 16.). What is cross-site scripting? XSS cheat sheet. Veracode. [URL.](https://www.veracode.com/security/xss)

Tim, T. (2023, March 20). What is SQL injection? tutorial & examples: Web security academy. What is SQL Injection? Tutorial & Examples | Web Security Academy. [URL.](https://portswigger.net/web-security/sql-injection)

Jain, S. (2024, March 18). Client-side form handling with JavaScript – explained with example code. freeCodeCamp.org. [URL.](https://www.freecodecamp.org/news/form-validation-in-javascript/)

**References**

XSS- <https://www.veracode.com/security/xss>

SQL Injection - <https://portswigger.net/web-security/sql-injection>

Form Validation - <https://www.freecodecamp.org/news/form-validation-in-javascript/>