

Information Security

Assignment 4 Report

Cross-Site Scripting

Group Members:

Mehmood Amjad

19I-0472 CS-F

Muhammad

19I-0561 CS-F

Danyal Faheem

19I-2014 CS-F

Table of Contents

1. JavaScript	3
1.1 Vulnerability	3
1.2 Secure	4
1.2.1 Input Validation	4
1.2.2 Input Sanitization	4
2. PHP	5
2.1 Vulnerability	5
2.2 Secure	6
2.2.1 Input Validation	6
2.2.2 Input Sanitization	7

1. JavaScript

1.1 Vulnerability

We first made a web page using simple username and password input. The UI of the web page can be seen in Figure 1.

UserName

432

Password

....

Submit

View Users

Name: 123 Password: 321

Name: 432 Password: 1234

Figure 1: UI of JS webpage

It has a simple input field and option to view the entered data later on. There is no sanitization or input validation.

Note: Modern JavaScript Browsers do not accept `<script>` tags as they sanitize it itself. For that reason, we will be showing other HTML tags here.

Now if we put in a HTML tag instead lets see the result in Figure 2.

UserName

<h1>Hello</h1>

Password

...

Submit

View Users

Name:

Hello

Password: 123

Figure 2: Output for HTML tag in input

Clearly, the webpage has a XSS vulnerability as it parsed the tag instead of the string.

1.2 Secure

To make the website secure, we utilized two methods. The two methods are input validation and sanitizing input.

1.2.1 Input Validation

For input validation, we just checked if there were any HTML tags present in the input. This was done using a regular expression that would look for both opening and closing tags.

The code can be seen as below:

```
// Make JSON object of data
user = { "userid": x['userid'].value, "password": x['pwd'].value };
// Regex to find all HTML tags
const regex = /<\/?[\w\d]+>/gi;

// If HTML Tags present, raise error
if (regex.test(user['userid']) || regex.test(user['password'])) {
    alert("HTML Tags found in input! Please Try again");
}
```

Now, let's try that input again. The result can be seen in Figure 3.

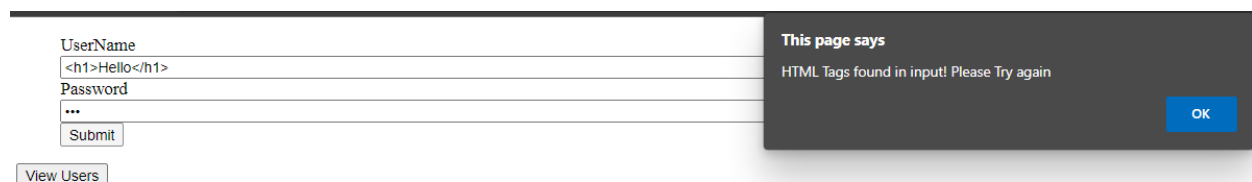


Figure 3: Input Validation Results

The HTML tags were found and it gave an error for invalid input. The input was asked to be input again.

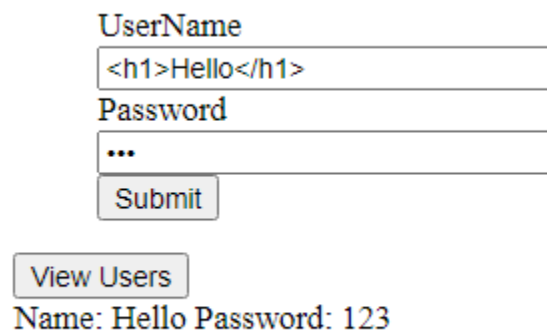
1.2.2 Input Sanitization

For Input Sanitization, a less orthodox method was used here and we would parse all the HTML tags inside the input and remove them all from the input altogether. This was also done using regular expressions that would look for all opening and closing HTML tags.

The code snippet can be seen below.

```
user['userid'] = user['userid'].replace(/<\/?[\w\d]+>/gi, "");  
user['password'] = user['password'].replace(/<\/?[\w\d]+>/gi, "");  
// Add to users list  
users.push(user)
```

Now let's try that input again. The results can be seen in Figure 4.



UserName
<h1>Hello</h1>
Password
...
Submit
View Users
Name: Hello Password: 123

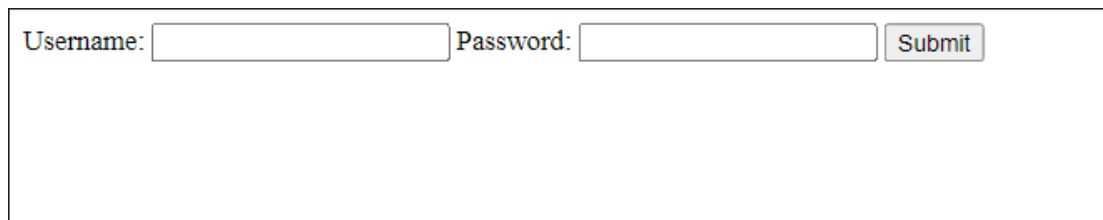
Figure 4: Input Sanitization Output results

The input was accepted however, the HTML tags were removed and only the text in between was kept.

2. PHP

2.1 Vulnerability

We first made a web page using simple username and password input. The UI of the web page can be seen in Figure 5.



Username: Password: Submit

Figure 5: PHP Web Page UI

It has a simple input field that then displays the data entered. There is no sanitization or input validation.

Now if we put in a HTML tag instead lets see the result in Figure 6.



Figure 6: Vulnerable Page output

As the alert was generated, it means the script was executed successfully.

2.2 Secure

To make the website secure, we utilized two methods. The two methods are input validation and sanitizing input.

2.2.1 Input Validation

For input validation, we simply checked if there was an opening <script> or a closing </script> tag present in the input, then we would prompt the user for invalid input.

The code snippet can be seen below.

```
// Trim to remove whitespaces
$username = trim($_POST['name']);
$password = trim($_POST['pass']);
// Look for script tag in username
if (strpos($username, "<script>") !== false || strpos($username,
"<\script>") !== false ) {
    echo "Script tag found in the username input, please try again";
}
```

```
// Look for password tag in username
elseif (strpos($password, "<script>") !== false || strpos($password,
"<\script>") !== false ) {
    echo "Script tag found in the password input, please try again";
}
```

Now, let's try that input again. The result can be seen in Figure 7.

Username: Password:

Script tag found in the username input, please try again

Figure 7: Input Validation Results

The aforementioned tags were found and it gave an error for invalid input.

2.2.2 Input Sanitization

For Input Sanitization, a less orthodox method was used here and we would parse all the HTML tags inside the input and remove them all from the input altogether. This was done using an inbuilt PHP function `strip_tags()` that would remove all the tags.

The code snippet can be seen below.

```
// Trim to remove whitespaces
$username = trim($_POST['name']);
$password = trim($_POST['pass']);
// Sanitize input nonetheless
$username = strip_tags($username);
$password = strip_tags($password);
```

Now, let's try that input again. The result can be seen in Figure 8.

Username: Password:

Welcome alert("hello")
Password 123

Figure 8: Input Sanitization Output results

The input was accepted however, the HTML tags were removed and only the text in between was kept.

Thank You!