Distributed Computing Assignment

COMP3008

XSS Report

By Aaron Gangemi

Student Number: 19447337

Contents

Introduction:

For this assignment, I have constructed using Visual Studio, 2 projects which are built as web services to demonstrate a reflective, persistent and DOM XSS attack. The first project implements each XSS attack whereas the second project demonstrates how to prevent each attack. The application is quite simple and includes features such as providing feedback about the site, providing information to the administrator, and creating a basic profile with only a username field.

According to Morris [1], the following XSS attacks are defined as follows:

1. Reflected XSS: is when the attacker inserts malicious code and the result is reflected to the victim's browser
2. Persistent XSS: is when the attacker injects malicious content into the application, its database or web server and is permanently stored by the application. The attack then affects all users when they load the webpage containing the malicious script.
3. DOM XSS: is when the attacker crafts malicious code. The browser then runs its legitimate code and the malicious code is then executed. The DOM XSS attack requires the client side to be vulnerable to attacks.

Directory Structure:

The folder containing the implementation for each attack is called **"XSS"**.

The folder containing the implementation to prevent each attack is called **"XSS_Application_Fixed"**.

XSS Attacks:

Reflective XSS:

The reflective XSS attack can be found in the file **"/XSS/Views/Home/Feedback.cshtml"**. This page can be accessed in the menu bar at the top of the page by clicking the "Feedback" tab.

How to cause a reflective XSS attack:

The **Feedback.cshtml** page contains an input field which allows the user to enter feedback and a submit button which allows the user to submit their feedback. As this file demonstrates an XSS attack, there is no validation on the input field. Once the user clicks submit, there feedback is displayed back to them.

To cause a reflective XSS on this application, the user is required to submit malicious JavaScript code which is then interpreted by the browser and displayed back to the user. An example of this would be to enter:

Example:

<script>alert("This is an example of a reflective XSS attack");</script>

The alert with text: "This is an example of a reflective XSS attack" will then be displayed back to the user when they click the submit button. Given the JavaScript text is displayed back to the user, the reflective XSS attack is successful.

How to prevent a reflective XSS attack:

The reflective XSS attack can be found in the file **"/XSS_Application_Fixed/Views/Home/Feedback.cshtml"**. This page can be accessed in the menu bar at the top of the page by clicking the "Feedback" tab. To prevent the attack, I have implemented regex which checks the input field upon submission for any special characters such as '<' or '>'. If

these are found, the program will continue to execute the given text, however if any special characters are found in the given string, they will be displayed using UTF-8 encoding. This has been implemented using the “encodeURIComponent()” function, which escapes special characters. An example of this is if I type the text:

Example:

<script>alert(“Hello World”);</script>

The following result will appear in the displayed string:

%3Cscript%3Ealert(%E2%80%9CHello%20World%E2%80%9D)%3B%3C%2Fscript%3E

As seen above, special characters have been escaped out of the given string, which makes the malicious JavaScript code impossible to execute on the client side.

Persistent XSS:

The persistent XSS attack implementation can be found in the files **“/XSS/Views/Home/TellAdmin.cshtml”** and **“/XSS/Views/Home/Admin.cshtml”**. These pages can be accessed in the menu bar at the top of each interface.

How to cause a persistent XSS attack:

The **TellAdmin.cshtml** acts like an email program as it contains text fields which allow the user to enter a message with an associated subject describing what the message is about. Once typed in, the user clicks submit, and a post request is then performed to submit the subject and message to the web server. The subject and item are stored in a MessageItem object which is then stored in a list. The only validation on the server is checking that the subject text and message text are not empty. If either the subject or message is empty, the program will display an error back to the user

Example:

Subject: “TestXSS1”

Message: <script>alert(“I am malicious”);</script>

From here, to access the second part of the persistent XSS attack, the user is required to click on the admin tab in the menu bar. The user will be redirected to **Admin.cshtml**. This page allows the administrator/user to read any messages that have been sent from the tell admin page. The page contains a drop-down list and the administrator/user will be required to select a subject. One of the subjects listed will be **“TestXSS1”**. Once the administrator/user clicks the subject, the associated message will appear below. However, in the above case, an alert will occur with the text “I am malicious”. This is caused as there is no validation on either the client or server side, and then the malicious script is loaded to the client side successfully.

How to prevent a persistent XSS attack:

The implementation to prevent the persistent XSS attack can be found in **“XSS_Application_Fixed/Controllers/WebController.cs”**. This file performs a range of validation before the message is added to the message list. The web controller performs the following checks:

1. If either the subject or message is null or empty
2. If any character of the subject and message string is a special character (not a letter or digit)

If the program finds either of these conditions to be true, it will throw an exception back to the TellAdmin.cshtml page and will never be displayed in the Admin.cshtml page.

Therefore, if the item is never submitted to the message list due to failed validation of each string, the persistent XSS attack has been prevented as malicious JavaScript will never be executed.

DOM XSS:

The DOM XSS attack implementation can be found in the file: **"/XSS/Views/Home/CreateProfile.cshtml"**. This page can be accessed in the menu bar at the top of each interface.

How to cause a DOM XSS attack:

The **CreateProfile.cshtml** page contains an input field for the user to enter their desired profile name. As the user is typing their profile name, the name will be reflected to them below as they are typing. This has been completed using the document.getElementByID().innerHTML function. If the user was to type malicious HTML code, then it would be reflected to the user using the innerHTML.

Example:

<button onclick=alert("I am a malicious button");>CLICK ME</button>

If the user types the above code in, the alert "I am a malicious button" will appear in the alert space. This simple example utilizes the DOM model in JavaScript to perform a successful DOM XSS attack on the web application. This code is then executed on the client side like normal HTML, and a button will appear that should not be there.

How to prevent a DOM XSS attack

The implementation to prevent the DOM XSS attack is stored in **"XSS_Application_Fixed/Views/Home/CreateProfile.cshtml"**. To prevent the DOM XSS attack, I have once again utilized regex, which checks that each character typed is not a special character. If the program recognizes a special character typed, it will display an alert stating that the profile name contains invalid characters and will then clear the text field so that the user can not continue to type or execute the malicious script. By doing this, no DOM XSS attack can be performed.

References:

[1] E. Morris, "Vulnerability Management", *3 Types of Cross-site Scripting (XSS) Attacks and How to Deal with Them*, 2019.