

**XSS VULNERABILITY WEB APPLICATION LAB**

**REPORT**



     By

**1.Dnyaneshwar shelke(B46)**

Email: [dnyaneshwarshelke0109@gmail.com](mailto:dnyaneshwarshelke0109@gmail.com)

Github link: <https://github.com/gitprogamforvenom?tab=repositories>

**2.Aarav Gaur(B46)er(**

Email: [aaravgaur759@gmail.com](mailto:aaravgaur759@gmail.com)

Github link: <https://github.com/desmond3546?tab=repositories>

)gig

Table of Contents

Xss vulnerabilities report1

1.introduction2

2.Objective3

3. Requirements2

4. ExeutiveSummary3

5.projectScope2

6.Methodologies3

1.Labsetup9

a.Web Application Setup: 9

b. Introduced Vulnerability11

2.Penetration testing23

2. 1.Method 1 (Automated Testing (if using Burp Professional )23

2.2 .Method 2 (Automation tool using OWSEP zap )28

3. xss scanning report 31

4. Severity Level35

5.proof Concept 36

6. MItigation 38

7.Conclusion38

**xss vulnerability report**

# Introduction

XSS is a security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal information, hijack sessions, or perform other malicious actions, compromising the security of web applications.

# Objective

The objective of this assessment was to identify Cross-Site Scripting (XSS) vulnerabilities within the target application, focusing on areas where user input is not properly sanitized or validated. The goal was to test various input points such as form fields, query parameters, and cookies for potential XSS flaws by injecting common malicious payloads. The testing aimed to determine whether the application reflects or stores user input in a way that allows the execution of arbitrary scripts in the browser, thereby posing risks like session hijacking, data theft, or user impersonation. Ultimately, the objective was to highlight these vulnerabilities and provide actionable recommendations to mitigate the associated security risks..

# Requirements

* **Burp Suite Installation**: A working installation of Burp Suite (Community or Pro version) to intercept and analyze web traffic.
* **Browser Configuration**: The browser must be configured to route traffic through Burp’s proxy for interception.
* **Target Application**: A web application or website with accessible input fields (e.g., search boxes, forms, or URL parameters).
* **XSS Payloads**: A set of common and advanced XSS payloads to test various attack vectors.
* **Burp Suite Tools**: Use of Burp's Proxy, Repeater, and Intruder tools for manual and automated testing of user input points.
* **Knowledge of XSS Types**: Understanding of reflected, stored, and DOM-based XSS to identify vulnerabilities effectively.
* **HTTPS Interception**: If the application uses HTTPS, Burp's SSL certificate must be installed in the browser for traffic decryption.

# ExecutiveSummary

This report presents the findings from testing a web application for Cross-Site Scripting (XSS) vulnerabilities. Using a combination of manual and automated testing techniques, including Burp Suite, various user input fields were analyzed to detect improper handling of input data. XSS payloads were injected into identified points such as search bars, forms, and URL parameters. The testing uncovered multiple instances of reflected and stored XSS vulnerabilities, where malicious scripts were executed in the user's browser, leading to potential security risks such as session theft or unauthorized data access. Recommendations for mitigation include implementing input validation, proper output encoding, and applying a robust Content Security Policy (CSP) to prevent exploitation. Addressing these vulnerabilities is vital for ensuring the safety of users interacting with the application..

# Recommendations

To mitigate XSS vulnerabilities, it is essential to implement strong input validation and output encoding across the application. All user inputs should be sanitized to reject any potentially dangerous characters or scripts, such as <, >, " and '. It is also important to use context-specific encoding, ensuring that data is properly encoded for HTML, JavaScript, or URL contexts to prevent execution of injected scripts. Additionally, implementing a strict Content Security Policy (CSP) can significantly reduce the risk by restricting the sources from which scripts can be loaded. Regular security testing, including the use of automated tools like Burp Suite, should be conducted to identify and address vulnerabilities. Developer education on secure coding practices is critical to ensure that XSS risks are minimized in future releases. By addressing these recommendations, the application’s security can be strengthened, protecting users from potential exploits.

# ProjectScope

|  |  |
| --- | --- |
| **ProjectName** | Cross-Site Scripting (XSS) vulnerabilities. |
| **Description** | During my assessment of the NFT platform’s user profiles, I identified a significant security concern related to Cross-Site Scripting (XSS). This vulnerability allows the injection of malicious scripts into user profile fields, specifically the social media handles. |
| **Scope** | http://localhost/demo |
| **Credentials** | NA |
| **TestScope** | Web application PenetrationTest |

1. **Methodologies**

The methodology for testing XSS vulnerabilities involves several key steps, starting with identifying input points within the web application that accept user data, such as forms, URL parameters, and cookies. Burp Suite is used to intercept and analyze these inputs by capturing HTTP/HTTPS traffic through its Proxy feature. Once potential input fields are identified, common XSS payloads are injected to test for vulnerability, utilizing Burp Repeater for manual testing and Burp Intruder for automated testing across multiple inputs. The responses are closely monitored to see if any payloads are reflected or executed in the browser, confirming the presence of an XSS vulnerability. Additionally, different XSS variants such as reflected, stored, and DOM-based XSS are tested to ensure comprehensive coverage. Finally, any vulnerabilities discovered are documented, and appropriate mitigation strategies are recommended to secure the application

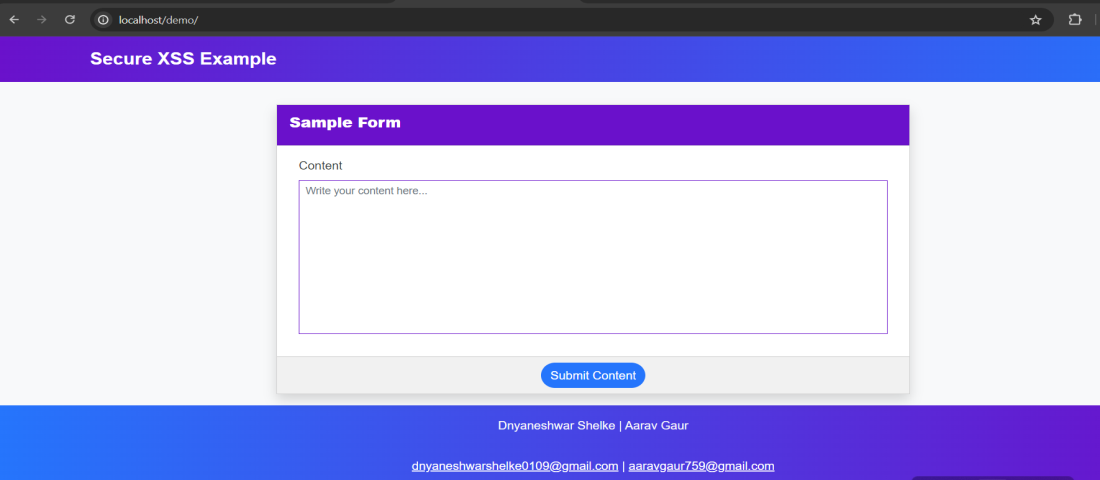
**1.Lab setup:**

**Objective:** Create a vulnerable web application with an input field where user input is not sanitized, enabling Cross-Site Scripting (XSS) vulnerabilities.

#### Web Application Setup:

We created a simple web application with the following features:

* **Input Fields:** A comment box where users can submit text

.

* **Backend:** A basic PHP setup (or any back-end technology that doesn't sanitize input) is used to process the data submitted by the user.

Github link:

Index.php (main file ):

<https://github.com/gitprogamforvenom/xss-project-/blob/main/demo/index.php>

<https://github.com/desmond3546/XSS-project/blob/main/index.php>

db\_connect.php (db file ) :

<https://github.com/gitprogamforvenom/xss-project-/blob/main/demo/db_connect.php>

<https://github.com/desmond3546/XSS-project/blob/main/db_connect.php>

**index.php**

**code :**

<?php

function escapeString($string){

return $string;

include 'db\_connect.php';

if ($\_SERVER["REQUEST\_METHOD"] === "POST") {

$content = $\_POST['dynamicContent'];

// Prepare the SQL statement to prevent SQL injection

$stmt = $conn->prepare("INSERT INTO contents (content, created\_at) VALUES (?, NOW())");

$stmt->bind\_param("s", $content);

if ($stmt->execute()) {

$success\_message = "Content saved successfully!";

} else {

$error\_message = "Error: " . $stmt->error;

}

$stmt->close();

}

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

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

<title>PHP - XSS Vulnerability Example</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/css/all.min.css" integrity="sha512-xh6O/CkQoPOWDdYTDqeRdPCVd1SpvCA9XXcUnZS2FmJNp1coAFzvtCN9BmamE+4aHK8yyUHUSCcJHgXloTyT2A==" crossorigin="anonymous" referrerpolicy="no-referrer" />

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.css" integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9Bv1WTRi" crossorigin="anonymous">

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

<script src="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/js/all.min.js" integrity="sha512-naukR7I+Nk6gp7p5TMA4ycgfxaZBJ7MO5iC3Fp6ySQyKFHOGfpkSZkYVWV5R7u7cfAicxanwYQ5D1e17EfJcMA==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>

<script src="https://code.jquery.com/jquery-3.6.1.js" integrity="sha256-3zlB5s2uwoUzrXK3BT7AX3FyvojsraNFxCc2vC/7pNI=" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-OERcA2EqjJCMA+/3y+gxIOqMEjwtxJY7qPCqsdltbNJuaOe923+mo//f6V8Qbsw3" crossorigin="anonymous"></script>

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

<style>

/\* General Reset \*/

body {

background: #f8f9fa;

color: #343a40;

font-family: 'Roboto', sans-serif;

line-height: 1.6;

}

nav.navbar {

background: linear-gradient(to right, #6a11cb, #2575fc);

}

nav.navbar .navbar-brand {

font-size: 1.5rem;

font-weight: bold;

}

nav.navbar .navbar-brand:hover {

color: #d4d4d4;

}

footer {

background: linear-gradient(to right, #2575fc, #6a11cb);

}

footer div {

padding: 1rem;

}

footer a {

color: #ffffff;

font-weight: 500;

}

/\* Main Content \*/

#main-wrapper {

margin-top: 2rem;

}

.card {

border-radius: 15px;

}

.card-header {

background: #6a11cb;

color: #fff;

border-radius: 15px 15px 0 0;

}

.card-header .card-title {

font-size: 1.25rem;

font-weight: bold;

}

.form-control {

border: 1px solid #6a11cb;

border-radius: 10px;

}

.form-control:focus {

border-color: #2575fc;

box-shadow: 0 0 5px rgba(37, 117, 252, 0.6);

}

.btn-primary {

background: #2575fc;

border: none;

}

.btn-primary:hover {

background: #6a11cb;

}

.card-footer {

background: #f1f1f1;

border-radius: 0 0 15px 15px;

}

/\* Responsive Design \*/

@media (max-width: 576px) {

nav.navbar .navbar-brand {

font-size: 1.2rem;

}

.card-header .card-title {

font-size: 1rem;

}

footer div {

font-size: 0.85rem;

}

}

</style>

</head>

<body>

<main>

<nav class="navbar navbar-expand-lg navbar-dark">

<div class="container">

<a class="navbar-brand" href="./">Secure XSS Example</a>

</div>

</nav>

</nav>

<div id="main-wrapper">

<div class="container px-5 my-3" >

<script>

start\_loader()

</script>

<div class="mx-auto col-lg-8 col-md-10 col-sm-12 col-xs-12">

<?php if($\_SERVER['REQUEST\_METHOD'] == "POST" && isset($\_POST['dynamicContent'])): ?>

<div class="card mb-3 rounded-0 shadow">

<div class="card-header rounded-0">

<div class="card-title"><b>Submitted Content</b></div>

</div>

<div class="card-body rounded-0">

<div class="container-fluid">

<?php

echo $\_POST['dynamicContent'];

?>

</div>

</div>

</div>

<?php endif; ?>

<div class="card rounded-0 mb-3 shadow">

<div class="card-header rounded-0">

<div class="card-title"><b>Sample Form</b></div>

</div>

<div class="card-body rounded-0">

<div class="container-fluid">

<form id="sample-form" action="" method="POST">

<div class="mb-3">

<label for="dynamicContent" class="form-label fw-light">Content</label>

<textarea class="form-control form-control-sm rounded-0"

name="dynamicContent"

id="dynamicContent"

rows="10"

style="resize:none"

placeholder="Write your content here..."><?php

?></textarea>

</div>

</form>

</div>

</div>

<div class="card-footer text-center rounded-0">

<button class="btn btn-primary rounded-pill" form="sample-form">Submit Content</button>

</div>

</div>

</div>

</div>

</div>

<footer class="text-center text-light">

<div>Dnyaneshwar Shelke <span id="dt-year"></span> | Aarav Gaur</div>

<div><a href="mailto:dnyaneshwarshelke0109@gmail.com">dnyaneshwarshelke0109@gmail.com</a><a> | </a><a href="mailto:aaravgaur759@gmail.com">aaravgaur759@gmail.com</a></div>

<div></div>

</footer>

</main>

</body>

</html>

**db\_connect.php**

**code:**

<?php

$servername = "localhost";

$username = "root";

$password = "";

$database = "xss\_example";

// Create connection

$conn = new mysqli($servername, $username, $password, $database);

// Check connection

if ($conn->connect\_error) {

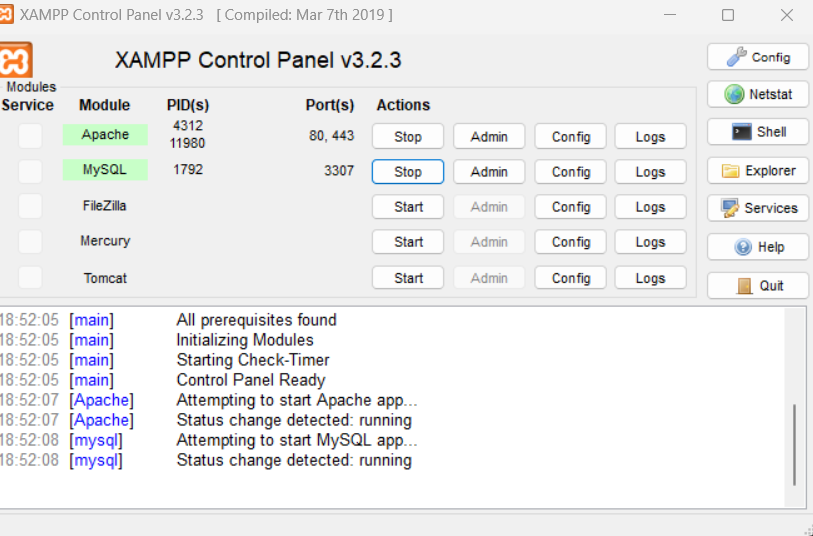
die("Connection failed: " . $conn->connect\_error);

}

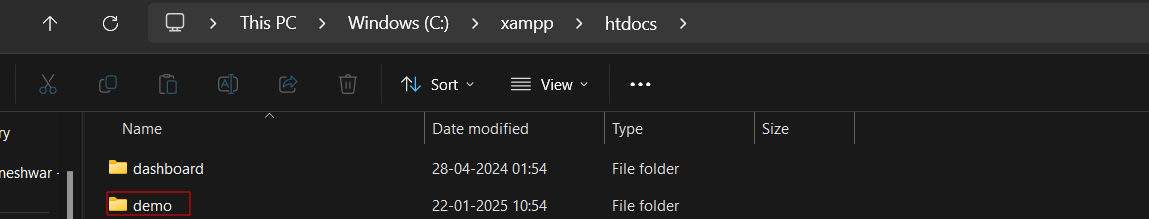
echo "Connected successfully";

?>

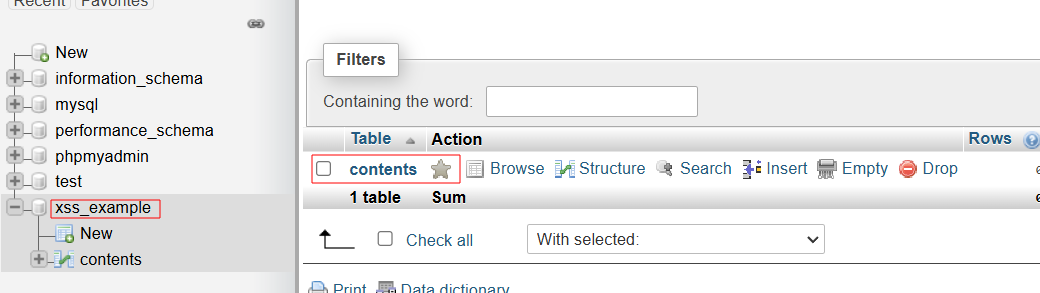
**Hosting web page :**



* **Download and Install XAMPP**: Get XAMPP from [apachefriends.org](https://www.apachefriends.org/), install it, and launch the XAMPP Control Panel.
* **Start Apache**: In the Control Panel, click "Start" next to Apache (start MySQL too if your site needs a database).
* **Prepare Website Files**: Ensure your HTML, CSS, JS, and other assets are ready.
* **Copy Files to htdocs**: Place your website folder inside the htdocs directory (e.g., C:\xampp\htdocs\demo).



* **Access Locally**: Open your browser and visit http://localhost/demo/.
* **Database Setup (if needed)**: Go to http://localhost/phpmyadmin/, create a database, and configure your site’s database connection.



* **Test Your Website**: Verify all pages and functionality work as expected in your browser.
* **Change Port (if needed)**: If port 80 is busy, update the port in httpd.conf to 8080 and access via http://localhost:8080/MyWebsite/.

#### b. Introduced Vulnerability:

We introduced a **Stored XSS** vulnerability by ensuring that the comment input is saved in a database (or a session) without sanitization. When another user accesses the page containing the stored comment, the malicious script is executed.

Example of malicious payload:

<script>alert('XSS')</script>

["onclick=prompt(8)><svg/onload=prompt(8)>"@x.y](mailto:%22onclick=prompt(8)%3e%3csvg/onload=prompt(8)%3e%22@x.y)

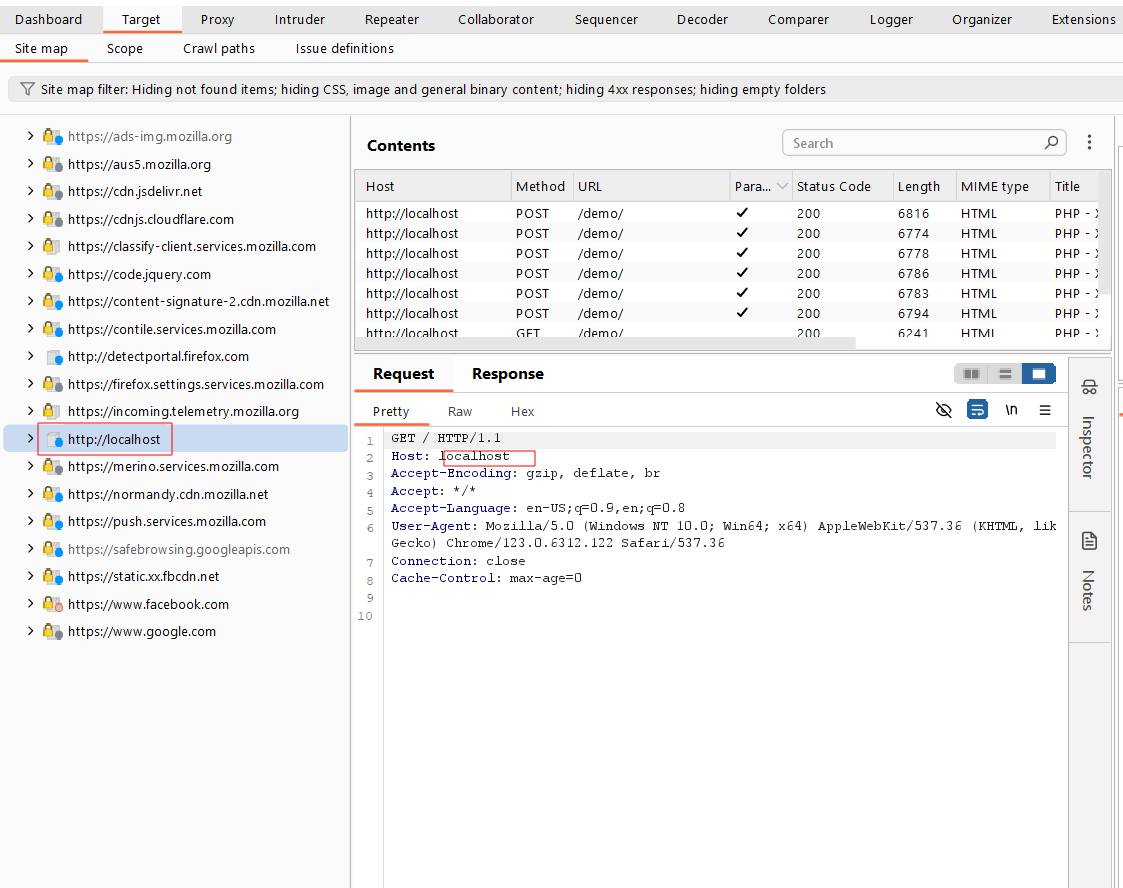
**2.Penetration Testing**

**2. 1 method 1 (burpsuite)**

**Objective:** Test the vulnerability by injecting malicious payloads and verifying its execution

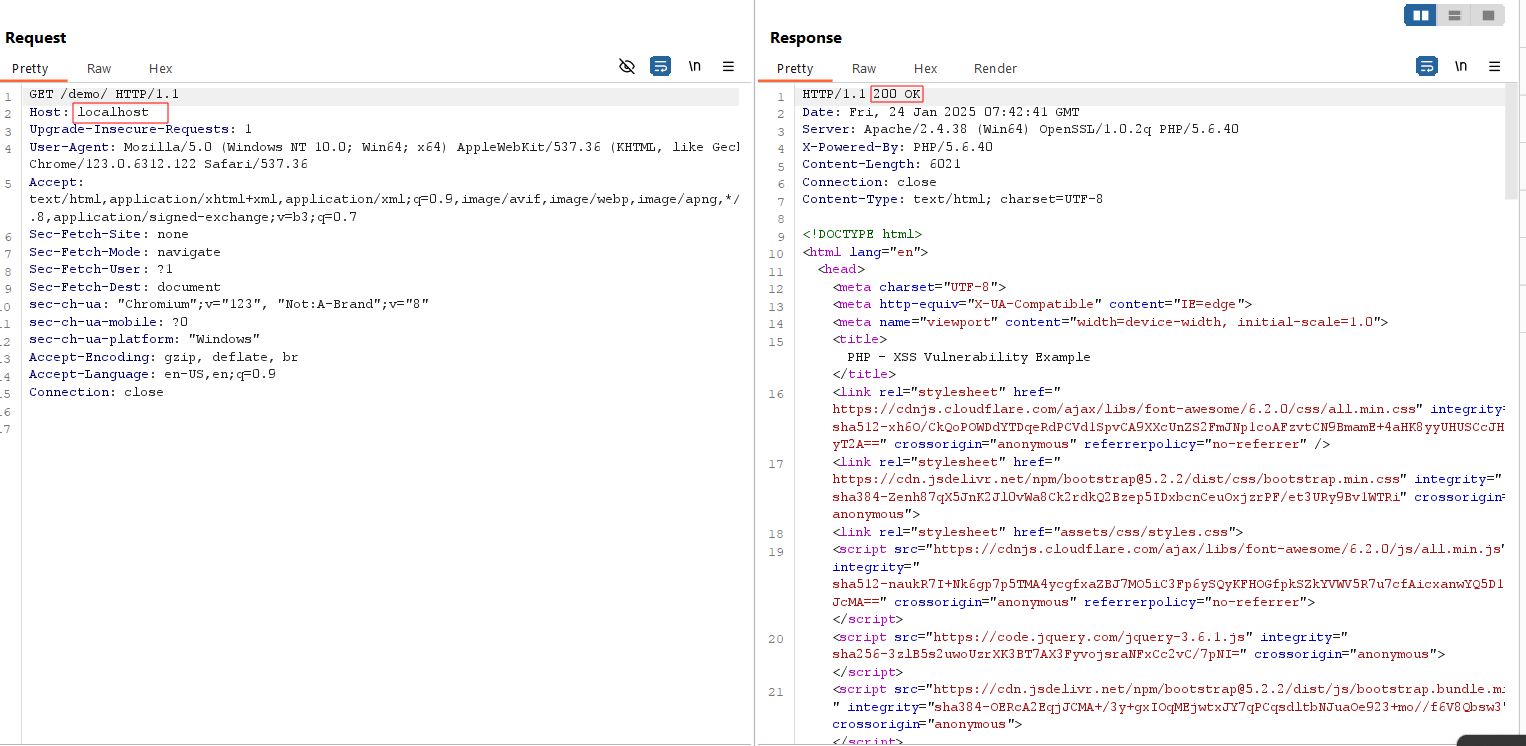
Automated Testing (if using Burp Professional)

1. **Configure Target Scope:**
   1. Navigate to the **Target** tab and set the target scope by right-clicking on the site and selecting **Add to Scope**.
   2. This ensures Burp focuses only on the desired application or pages.

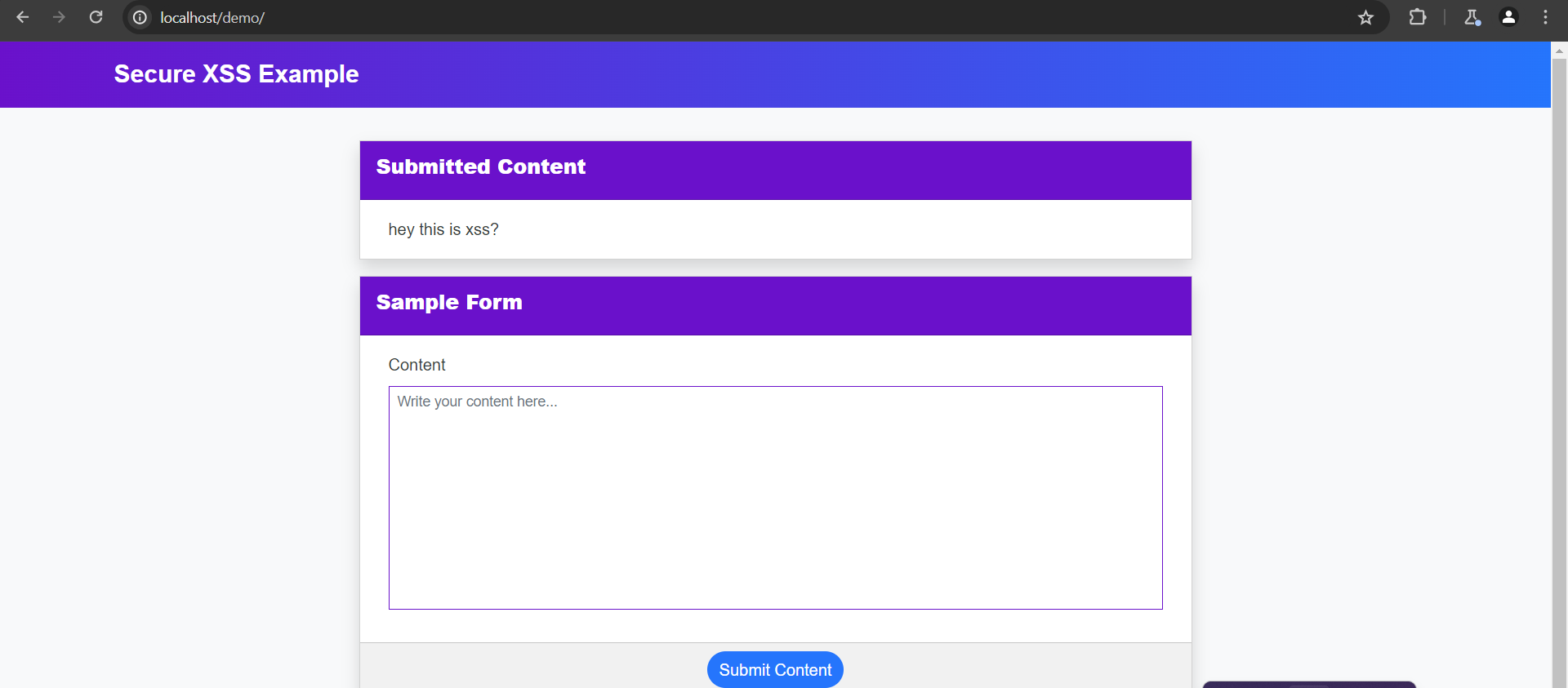


2.

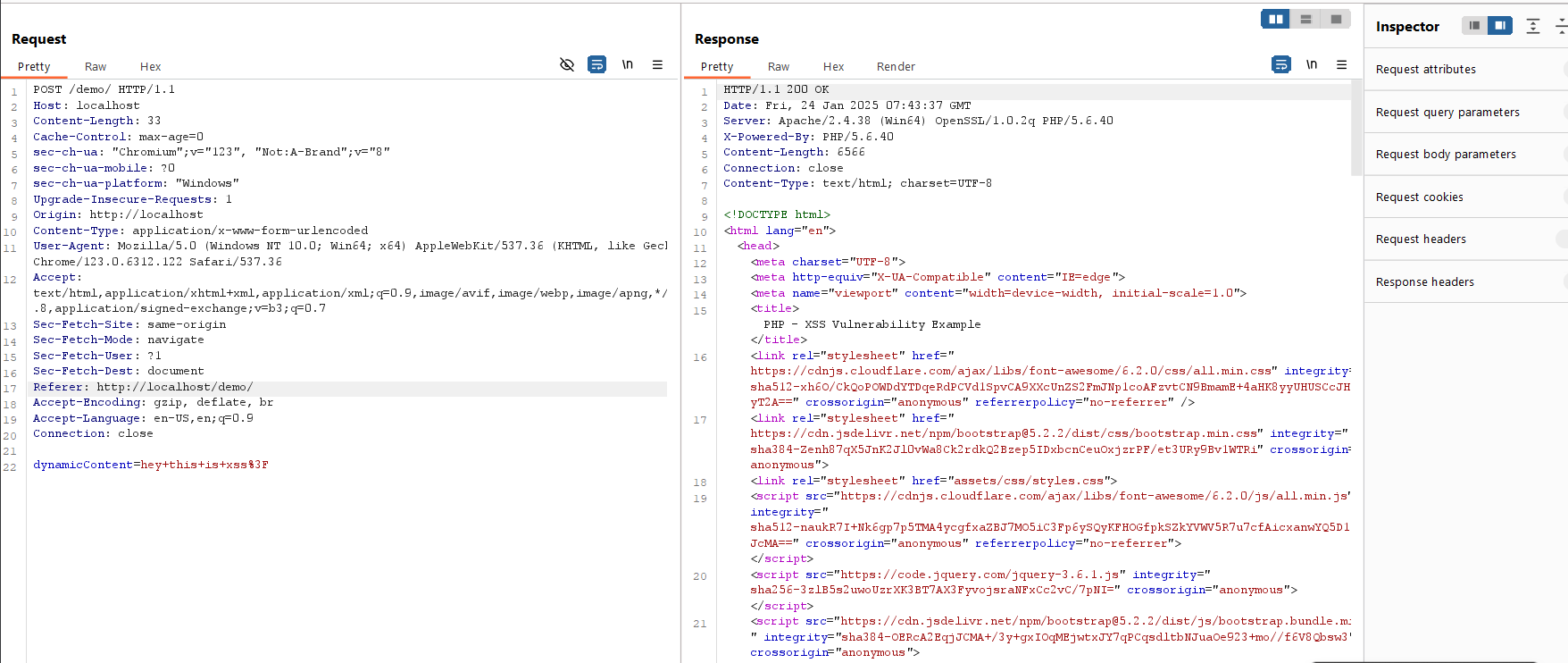
* Check captured traffic in the **HTTP history** tab (under **Proxy**) or the **Target** tab.
* Look for potential input points, especially areas where user inputs are reflected back in responses (e.g., query parameters in the URL or form inputs).



3.Submit a normal content below page



4. **Send Request to Intruder**: Right-click a captured request and select **“Send to Intruder”**.

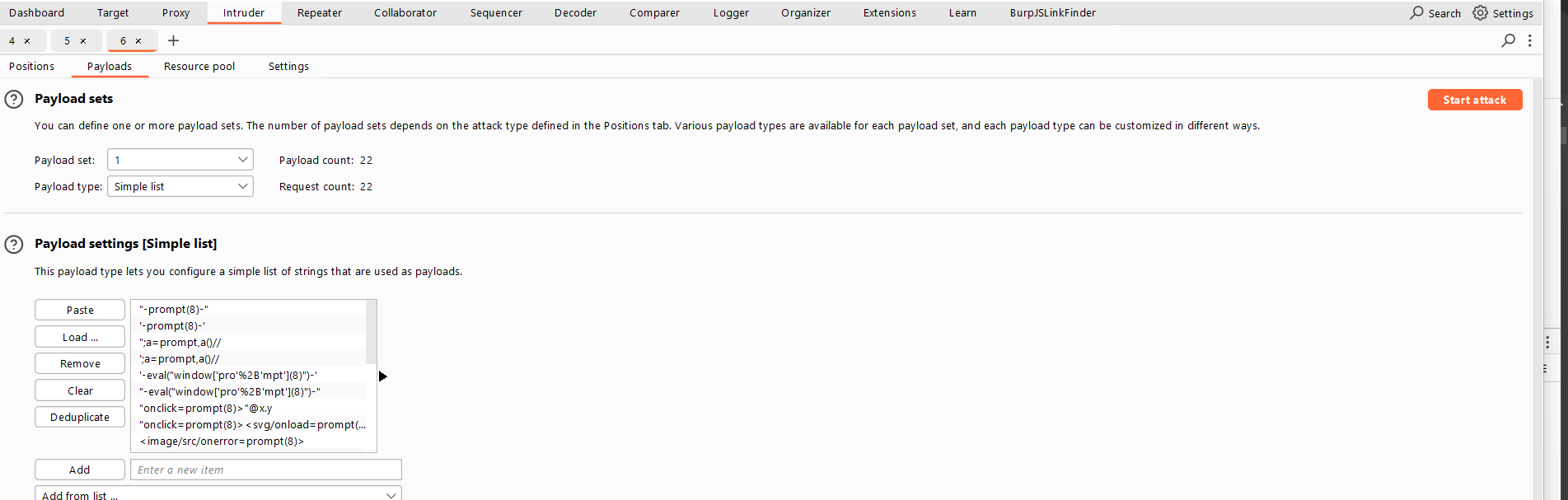


6.

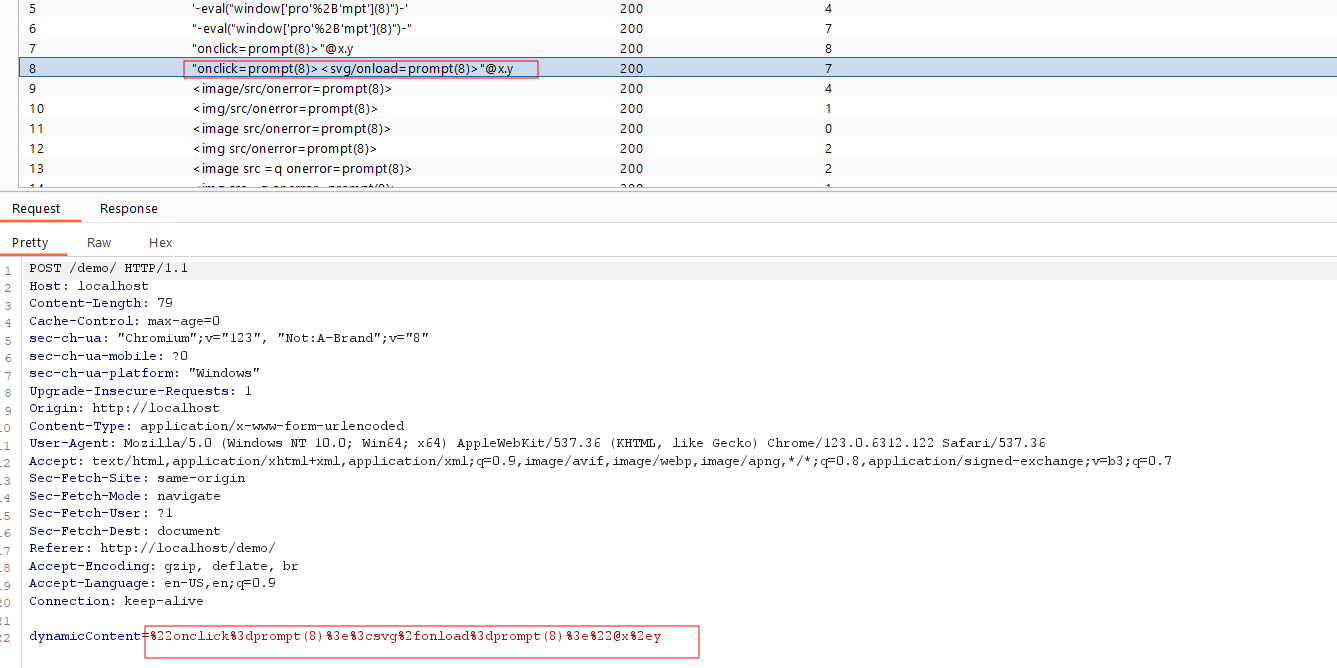
* **Send Request to Intruder**: Right-click a captured request and select **“Send to Intruder”**.
* **Load Payloads**:Use Burp’s built-in XSS payloads or import a custom payload list.

referencelink:<https://raw.githubusercontent.com/payloadbox/xss-payload-list/refs/heads/master/Intruder/xss-payload-list.txt>

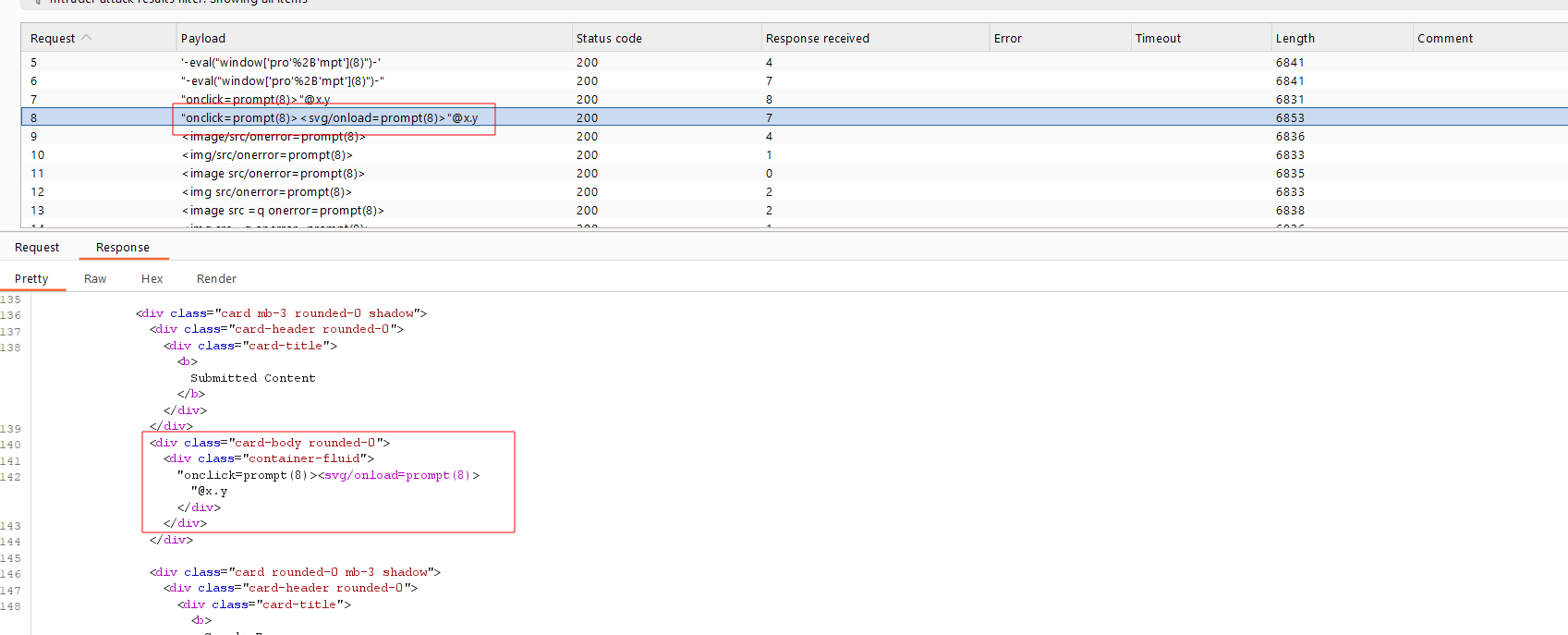
* **Start Attack**:Run the attack and analyze the responses for payload reflections or execution.



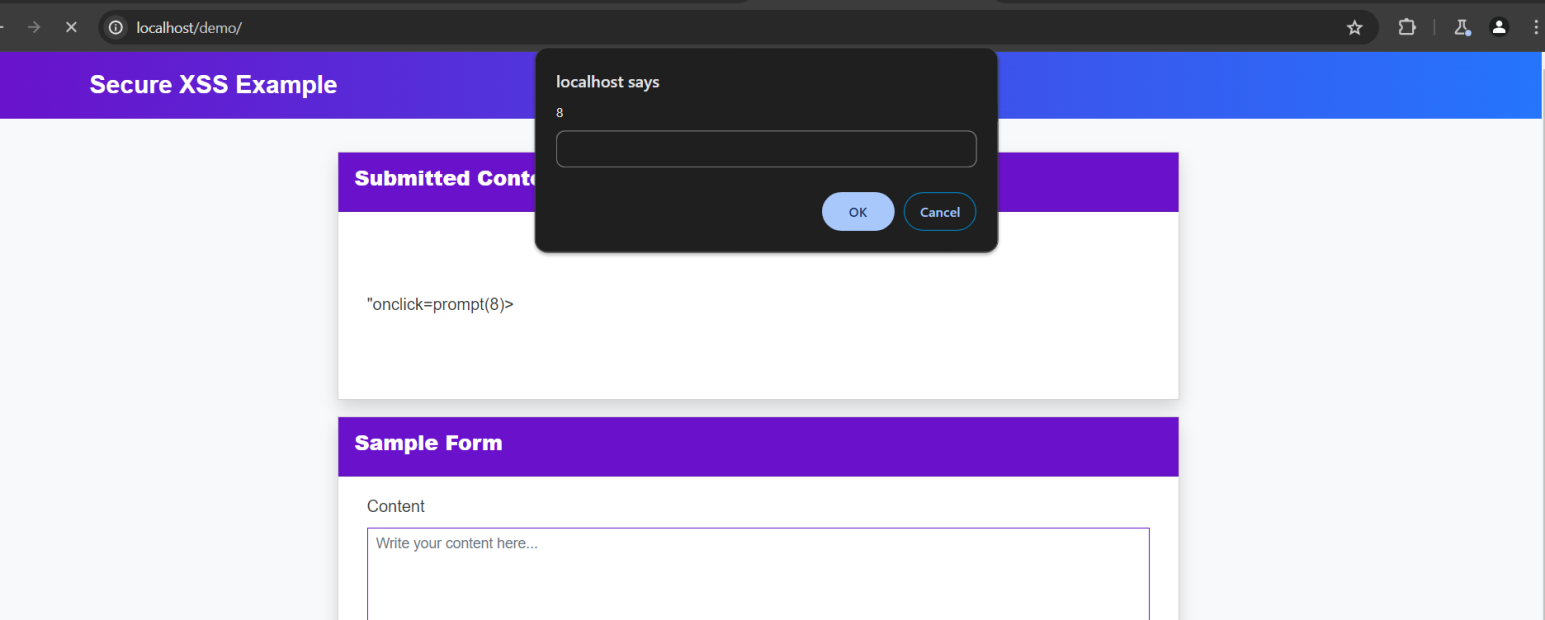
7.200 ok request payload : "onclick=prompt(8)><svg/onload=prompt(8)>"@x.y



8. In following img display response xss payload



9.copy response url and past browser finally find xss



**Video POC:**

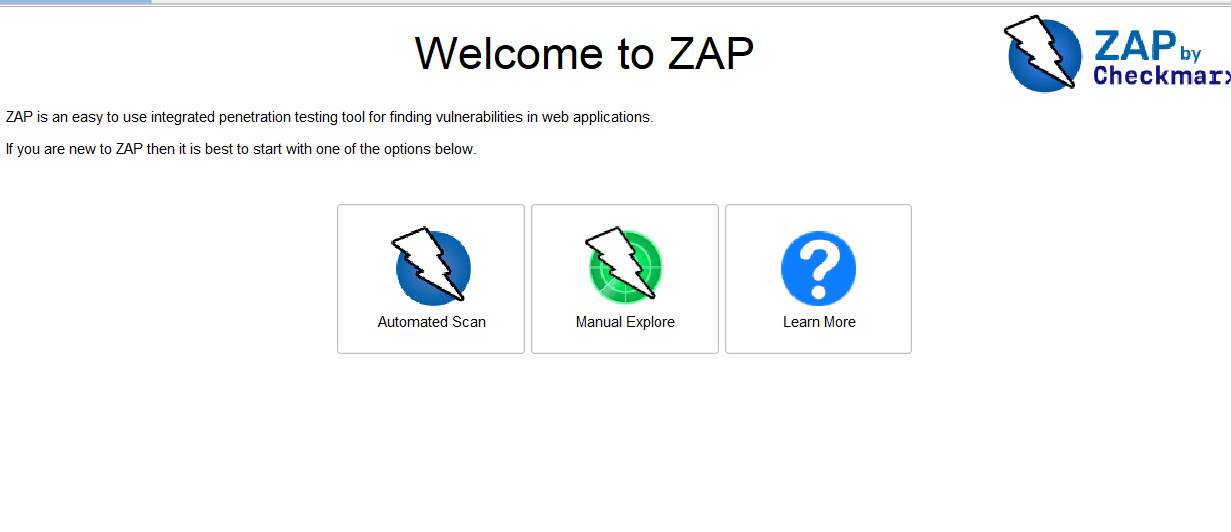
[**https://drive.google.com/file/d/1DtJhqgPeV9KurAW0P4fjmFPCIAK41VeP/view?usp=sharing**](https://drive.google.com/file/d/1DtJhqgPeV9KurAW0P4fjmFPCIAK41VeP/view?usp=sharing)

**2.2 method 2 (Owasp zap)**

1.

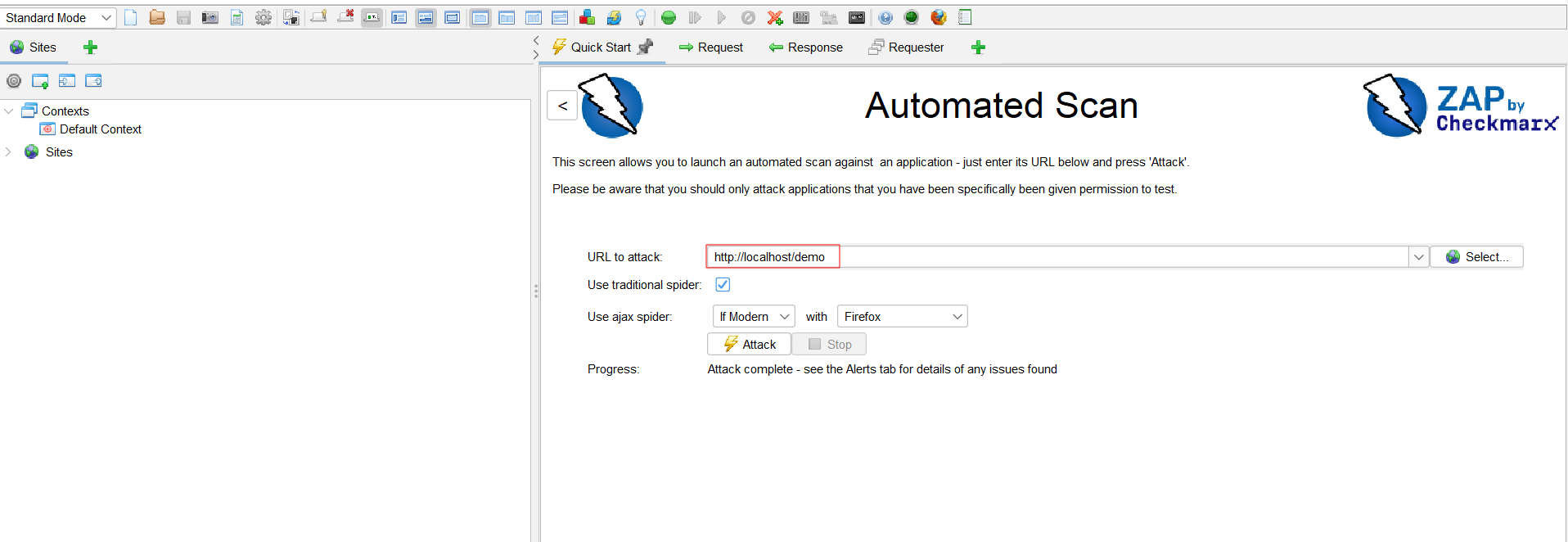
Launch OWASP ZAP and choose the appropriate mode:

* + **Safe mode**: No potentially harmful actions.
  + **Protected mode**: Some potentially harmful actions allowed.
  + **Attack mode**: Fully automated and aggressive scanning.



2.**Set Up the Target Application**

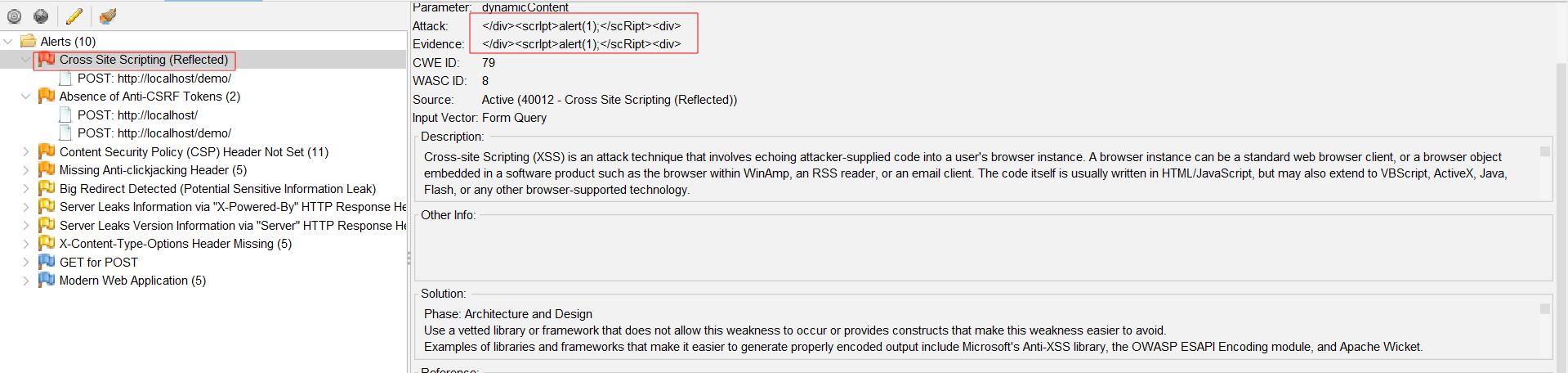
* Configure your browser to route traffic through OWASP ZAP’s proxy (default: localhost:8080).
* Open the target application in the browser to ensure traffic is captured by ZAP.
* Add the target URL to the **Scope**:
  + Go to **Sites** in the left-hand panel, right-click the target URL, and select **Include in Context** > **Default Context**.

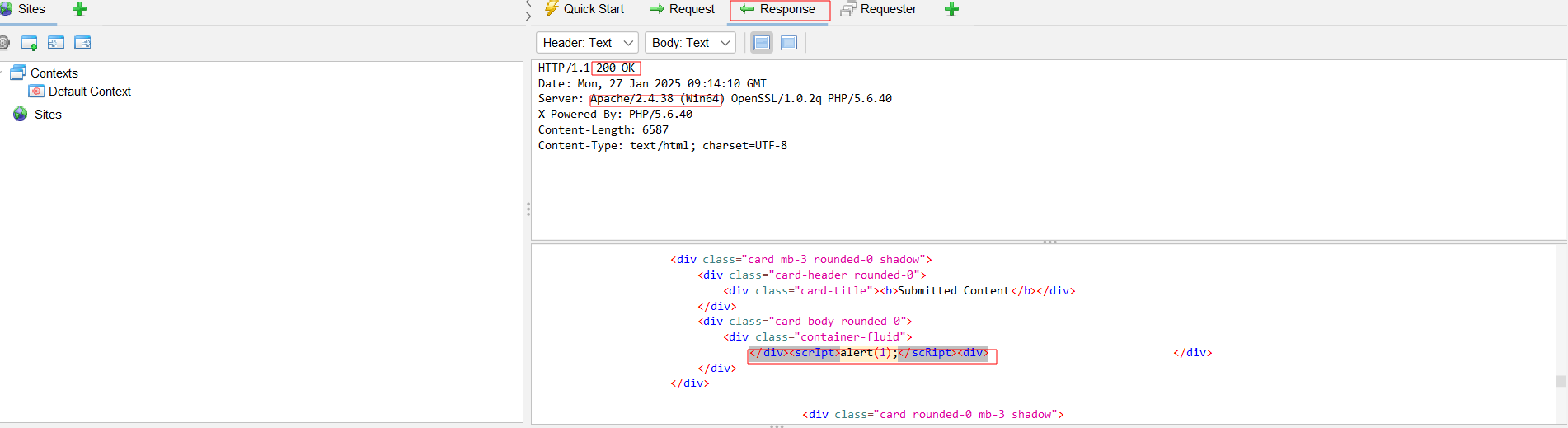


3. Analyze the Scan Results

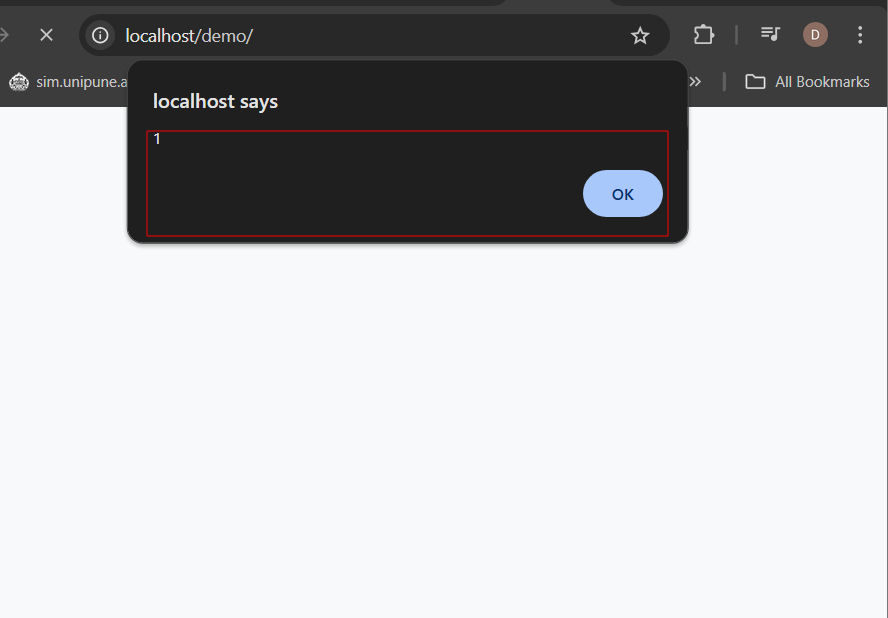
Once the scan completes, go to the **Alerts** tab.

Look for findings categorized as **Cross-Site Scripting**



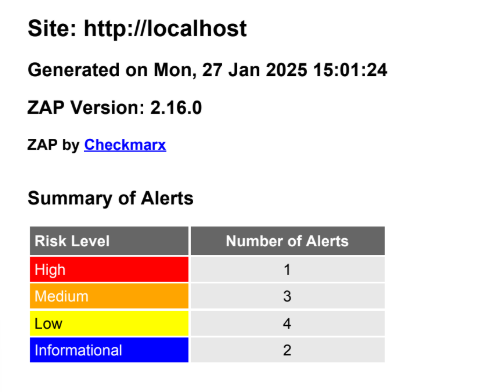
4. read a response 

5. copy a link and past web finally find xss

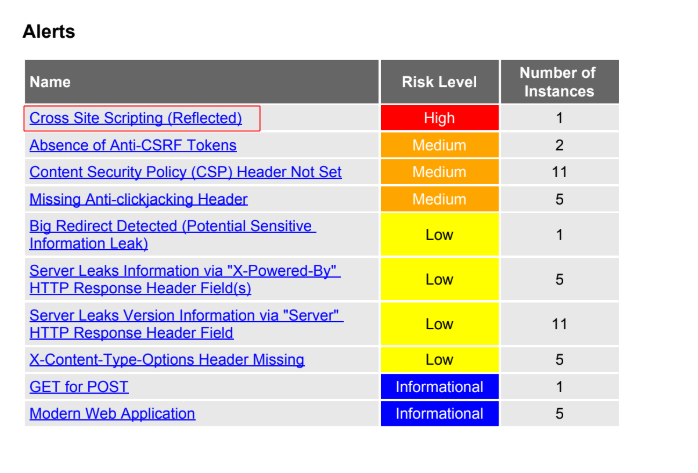


**3.Xss Scanning Report**

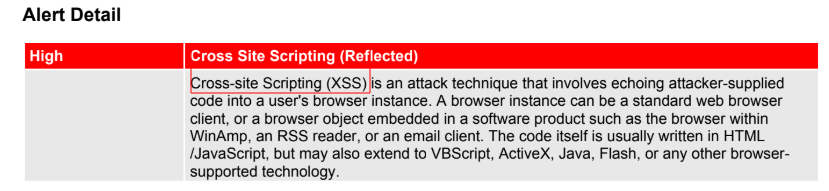
**1.**

****

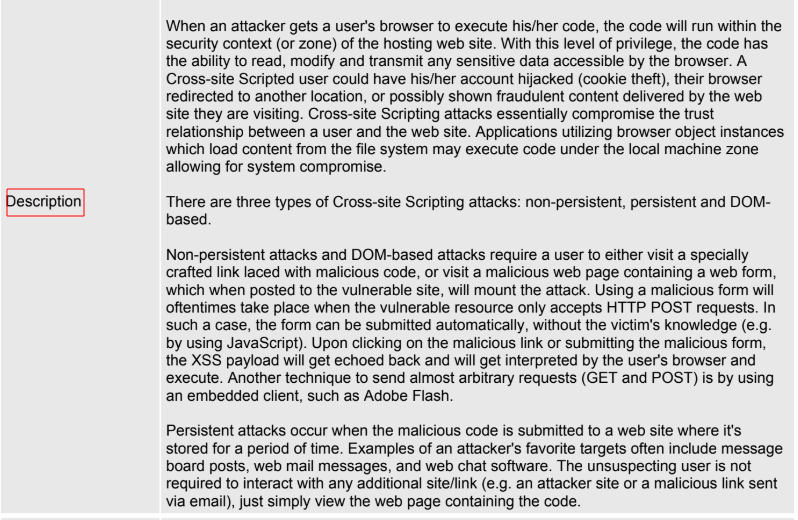
**2.**

****

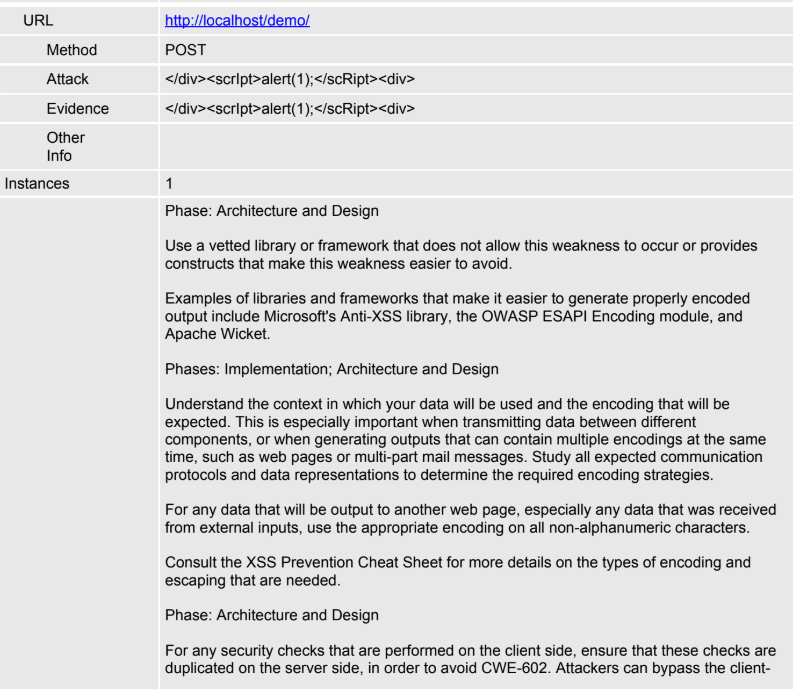
**3.**

****

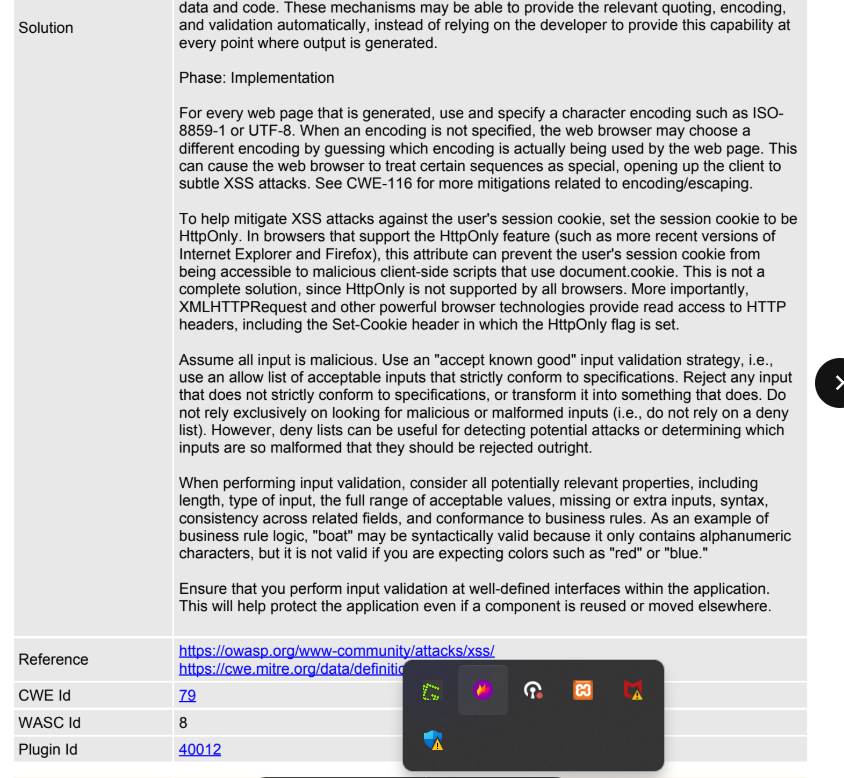
**4.**

****

**5.**

****

**6.**

****

**Video Poc (zap):** [**https://drive.google.com/file/d/1GUZzAhwGiVuE7jz6DRjajZhvIlyc1E4Z/view?usp=sharing**](https://drive.google.com/file/d/1GUZzAhwGiVuE7jz6DRjajZhvIlyc1E4Z/view?usp=sharing)

**Scanning report :** [**https://drive.google.com/file/d/1ct\_90DI\_NdHxef\_7Lnu\_I2uL1dAz-wsa/view?usp=sharing**](https://drive.google.com/file/d/1ct_90DI_NdHxef_7Lnu_I2uL1dAz-wsa/view?usp=sharing)

# 4.Severity Level

**Severity Level: High to Critical**

Stored XSS (Persistent XSS) is considered one of the most **severe** web security vulnerabilities. It is categorized as **High to Critical** on the **CVSS (Common Vulnerability Scoring System)** due to the following reasons:

* The malicious script is permanently stored in the application's database or backend.
* It is **automatically executed** whenever affected pages are loaded by users.
* Affects **multiple users** without requiring them to click on a malicious link.
* Can lead to **credential theft, session hijacking, malware distribution, and full account takeovers**.

| **Severity** | **CVSS Score** | **Impact** |
| --- | --- | --- |
| **High** | 7.0 - 8.9 | Multiple users affected, potential for account compromise. |
| **Critical** | 9.0 - 10.0 | Sensitive data theft, full system compromise, malware propagation. |

### ****Impact Assessment of Stored XSS****

#### ****1. Large-Scale User Data Theft****

* Attackers can inject malicious JavaScript into a comment section, profile bio, or any user-generated input field.
* When another user visits the affected page, the script **steals session tokens, cookies, or personal data** and sends them to the attacker.
* Can lead to **account takeovers, financial fraud, and identity theft**.

#### ****2. Full Session Hijacking****

* The attacker can steal **session cookies** and impersonate users, including administrators.
* This may grant **unauthorized access** to sensitive areas of a website or web application.

#### ****3. Defacement & Fake Content Injection****

* Attackers can modify the page content **permanently** by injecting fake messages, images, or forms.
* Users might see misleading information, fake promotions, or phishing pages disguised as the original site.

#### ****4. Malware Injection & Ransomware Attacks****

* The injected script can force users to **download and execute malicious files**.
* Attackers may distribute **ransomware, keyloggers, or spyware** through infected websites.

#### ****5. Automated Botnet Attacks & Worm Propagation****

* Stored XSS can be used to create self-replicating **XSS worms** that spread automatically to other users.
* Famous case: **Samy Worm (2005)** on MySpace, which infected over **1 million users in 24 hours**.

#### ****6. Phishing & Social Engineering****

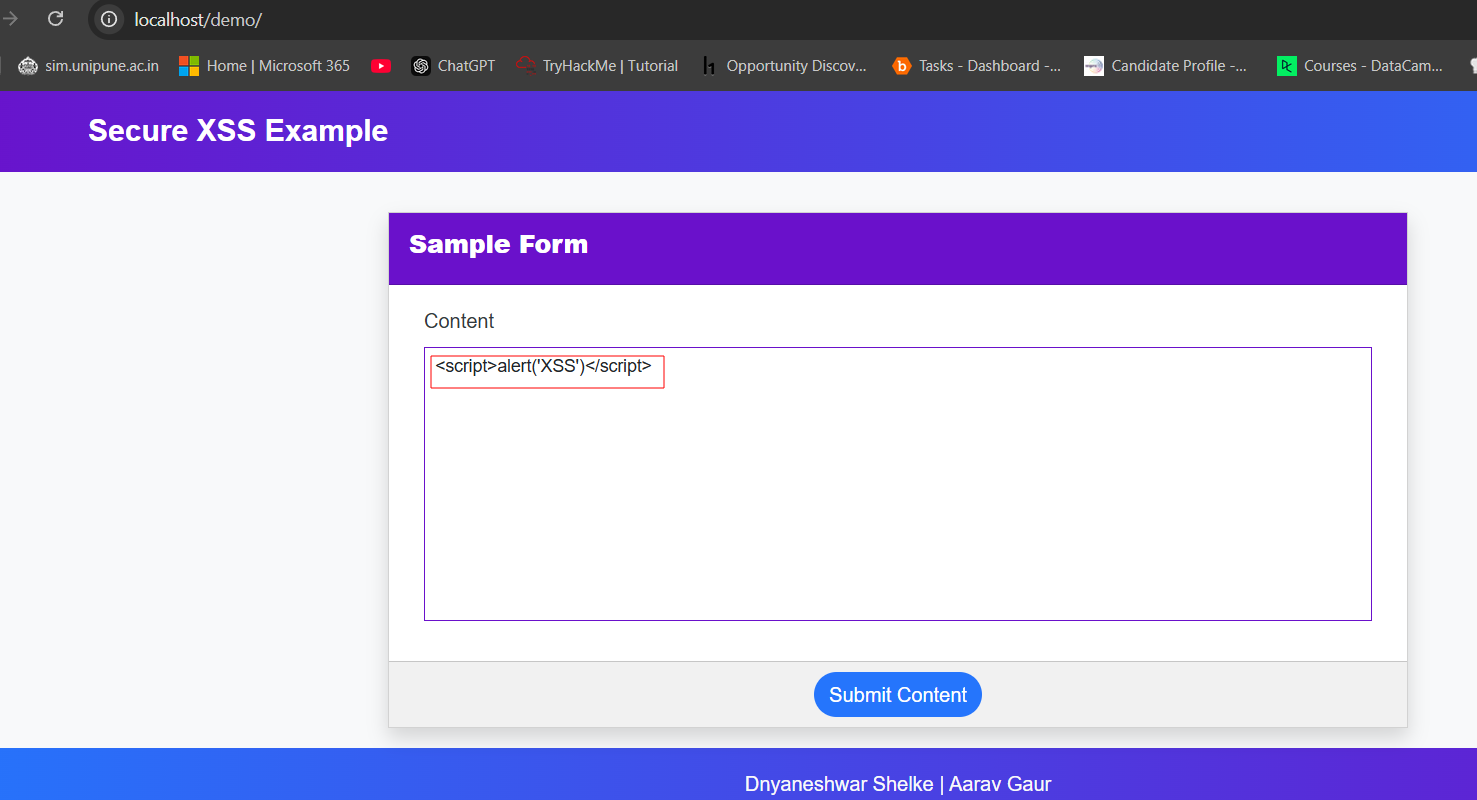
* Attackers can inject fake login forms to **harvest user credentials**.
* Users may be tricked into entering sensitive data, believing they are on a trusted site.

# 5.Proof of Concept (Poc)

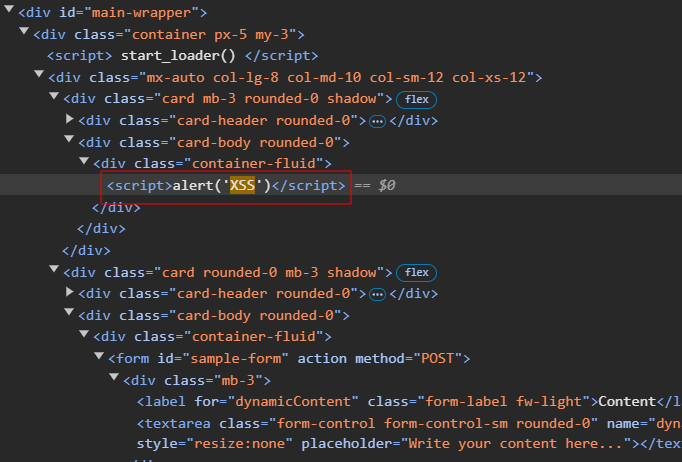
### ****1.Injecting the Malicious Script****

Post the following JavaScript payload into a comment field or any other input that gets stored and displayed to users:

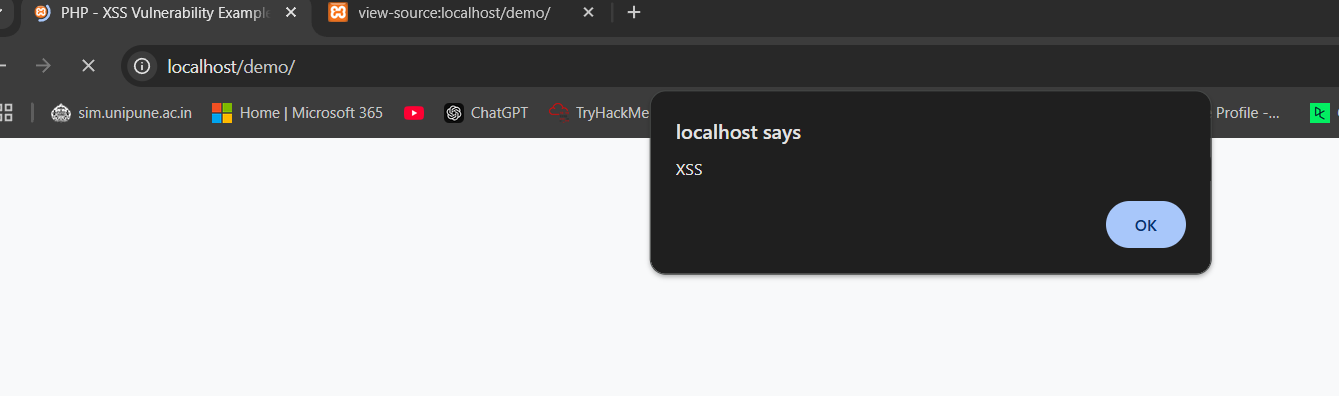
<script>alert('XSS')</script>

****

**2.Script Stored in the Web Page Source**

****

**3.XSS Alert Execution**

****

**Video poc:** [**https://drive.google.com/file/d/1e3P5\_FLMEfNuWp4nJNoSwk6jAlmC9K\_C/view?usp=sharing**](https://drive.google.com/file/d/1e3P5_FLMEfNuWp4nJNoSwk6jAlmC9K_C/view?usp=sharing)

# 6.Mitigation

## Input Validation and Sanitization

Validate and sanitize user inputs to ensure they don’t contain malicious scripts. Use libraries or built-in functions to escape special characters.

## Content Security Policy (CSP)

Implement a CSP to restrict the sources from which scripts can be loaded, significantly reducing the risk of XSS.

## Escaping Outputs

Ensure that all outputs are properly escaped in HTML, JavaScript, and other contexts to prevent the execution of injected scripts.

**7.Conclusion**

XSS is a powerful technique for exploiting web application vulnerabilities. By understanding and practicing the techniques discussed, you can effectively identify and mitigate these vulnerabilities. Set up your test environment and try out the XSS techniques to enhance your penetration testing skills.