Lab 1: Introduction to Web Technologies and Understanding HTTP and HTTPS

Overview

In this lab, you will delve into the foundational concepts of web technologies, specifically focusing on the HTTP and HTTPS protocols that enable seamless communication between clients (browsers) and servers. You will learn about the structure of web requests and responses, the significance of various HTTP methods, and the critical importance of securing web traffic through HTTPS. This knowledge will lay the groundwork for understanding more complex topics in web and database security.

Prerequisites

- Basic familiarity with web application concepts (HTML, CSS, JavaScript).
- Access to a modern web browser (e.g., Chrome, Firefox) and a text editor (e.g., VSCode, Notepad++).
- Understanding of client-server architecture and the basics of networking.

Lab Objectives

- Grasp the fundamental components of web technologies and their roles.
- Analyse the structure and functionality of HTTP and HTTPS.
- Understand the implications of HTTP status codes and headers.
- Learn how to inspect and analyse web traffic using browser tools.

Exercise Instructions

Exercise 1: Creating a Simple Web Page

1. Create a New HTML File:

- Open your text editor and create a new file named index.html.

2. Write HTML Code:

- Construct a basic web page that includes:
    - A title reflecting the content of the page.
    - A header that introduces the topic.
    - A paragraph explaining the importance of web technologies.
    - An image that visually represents web technologies (you can use a placeholder image).
    - A link to an external resource for further learning.

```html
<!DOCTYPE html>
<html lang="en">
<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial
scale=1.0">
<title>Introduction to Web Technologies</title>
<style>
body { font-family: Arial, sans-serif; margin: 20px; }
header { background-color: #f2f2f2; padding: 10px; }
img { max-width: 100%; height: auto; }
</style>
</head>
<body>
<header>

<h1>Understanding Web Technologies</h1>
</header>
<p>Web technologies are essential for creating and
maintaining dynamic websites that facilitate user
interaction.</p>
<img src="https://via.placeholder.com/600×300" alt="Web
[ Read 29 lines ]
```

3. Open in Browser:

- Save the index.html file and open it in your web browser to view the web page you created.

4. Reflection:

- Discuss what you learned about the structure of HTML documents. How do elements such as headers, paragraphs, and links contribute to a web page's content and usability?

In this exercise, I learned that **HTML provides the structural foundation of web pages**, where each element serves a distinct purpose:

- **Headers (`<h1>`–`<h6>`):** Organize content hierarchically and guide users through the page.
- **Paragraphs (`<p>`):** Present text clearly and improve readability.
- **Images (`<img>`):** Enhance understanding with visual representation.
- **Links (`<a>`):** Connect users to additional resources, enabling navigation beyond the page.

By creating this page, I also understood how **HTML elements combine to form a cohesive user experience**. Proper use of structure, content, and media improves usability, accessibility, and comprehension for visitors. Additionally, viewing the page in a browser reinforced the relationship between HTML code and how it renders visually.

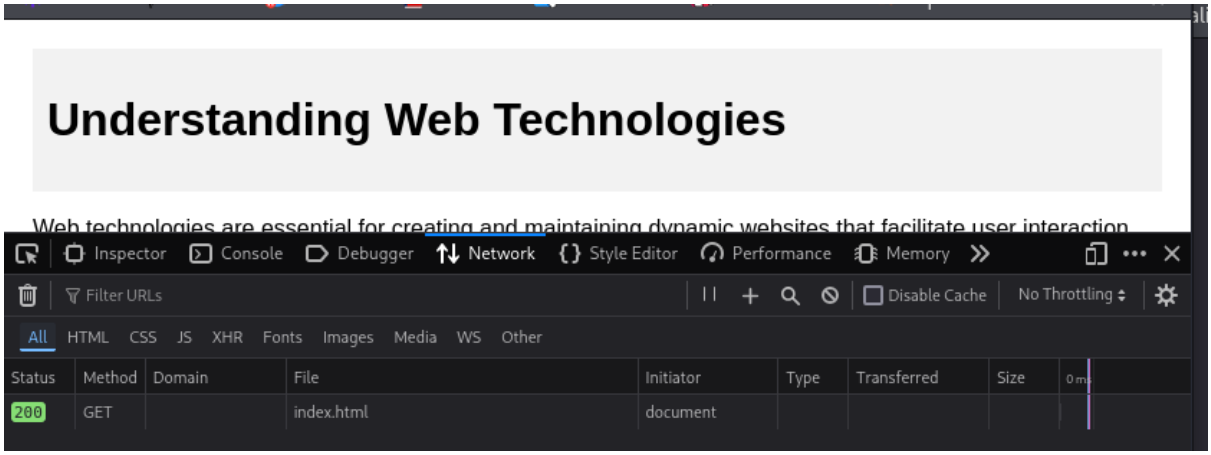
Exercise 2: Analysing HTTP Requests and Responses

1. Use Developer Tools:

- Open your browser's developer tools (F12 or right-click → Inspect) and navigate to the Network tab.

2. Refresh the Web Page:

- With the Network tab open, refresh your page to monitor the network requests being made.



3. Inspect HTTP Requests:

- Click on the first request for index.html and observe the following details:
  - Request URL: The address of the resource.
  - Request Method: Understand different methods (GET, POST, etc.).
  - Response Status Code: What does a status code of 200 OK mean?
  - Response Headers: Review headers like Content-Type, Cache Control, and Date.

4. Reflection:

- **Analyze the significance of each element. Why is it important to understand these components when developing or securing web applications? Discuss the potential vulnerabilities associated with improperly handled requests and responses.**

By analyzing HTTP requests and responses, I learned the following:

- **Request URL:** Identifies the exact resource the client is requesting; understanding URLs helps in routing, API calls, and avoiding broken links.
- **Request Method (GET, POST, etc.):** Determines how data is sent and handled by the server. Using the correct method is crucial for security (e.g., sensitive data should not be sent via GET).
- **Response Status Code:** Indicates whether the request succeeded (200 OK), failed (404 Not Found), or caused a server error (500). Knowing status codes helps debug issues and enforce proper error handling.
- **Response Headers:** Provide metadata about the response, such as content type and caching instructions. Misconfigured headers can lead to security vulnerabilities like XSS or sensitive data exposure.

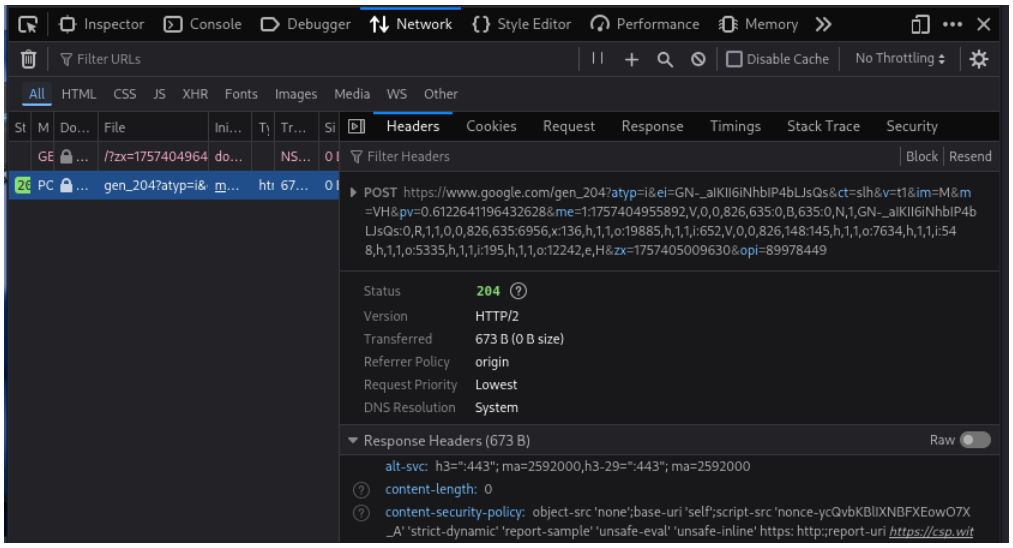
Exercise 3: Understanding HTTPS and Its Importance

1. Access a Secure Website:

- In your web browser, navigate to a website that uses HTTPS (e.g., https://www.google.com).

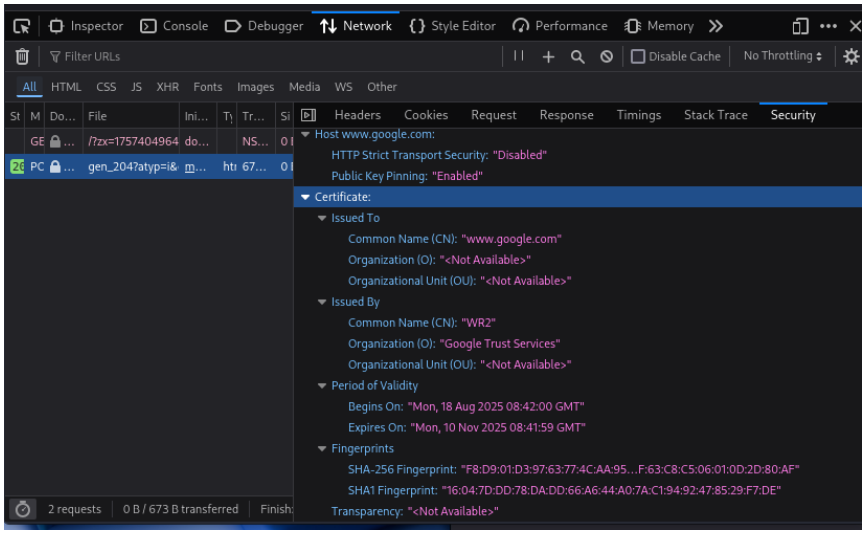
2. Inspect HTTPS Traffic:

- Open the developer tools and examine the network requests. Note how HTTPS requests are displayed differently compared to HTTP requests.



3. Explore the Security Information:

- Click on the padlock icon in the address bar. Investigate the security certificate details, including:
    - Certificate issuer
    - Validity period
    - Encryption methods used



4. Reflection:

- **Discuss the advantages of using HTTPS over HTTP, particularly in terms of data security and privacy. What protections does HTTPS provide against attacks such as eavesdropping and man-in-the-middle (MitM)?**

By exploring HTTPS, I learned that it provides **secure communication between clients and servers**. Key points include:

- **Encryption:** HTTPS encrypts data using TLS/SSL, preventing attackers from eavesdropping on sensitive information like passwords or credit card numbers.
- **Authentication:** Certificates verify the identity of the website, reducing the risk of connecting to malicious or fake sites.
- **Data integrity:** HTTPS ensures that data is not tampered with during transit, protecting against modification by attackers.
- **Protection against MitM attacks:** Since traffic is encrypted and certificates are verified, attackers cannot easily intercept or alter communications between the user and server.

Overall, using HTTPS significantly improves **privacy, trust, and security** for both users and web applications, making it a fundamental standard for modern web development.

Lab 2: Web Development Components and Security Threats

Overview

In this lab, you will explore the various components involved in web development, including front-end and back-end technologies. You will also identify and analyze common security threats associated with these components. Through practical exercises, you will gain hands-on experience in recognizing vulnerabilities and implementing secure coding practices.

Prerequisites

- Basic knowledge of HTML, CSS, and JavaScript.
- Familiarity with server-side programming languages (e.g., PHP, Node.js).
- Access to a web development environment (e.g., local server, cloud platform).

Lab Objectives

- Understand the key components of web development (front-end and back-end).
- Identify common security threats associated with web development components.
- Analyse the implications of these threats on web applications.
- Implement basic secure coding practices to mitigate vulnerabilities.

Exercise Instructions

Exercise 1: Exploring Front-End Development Components

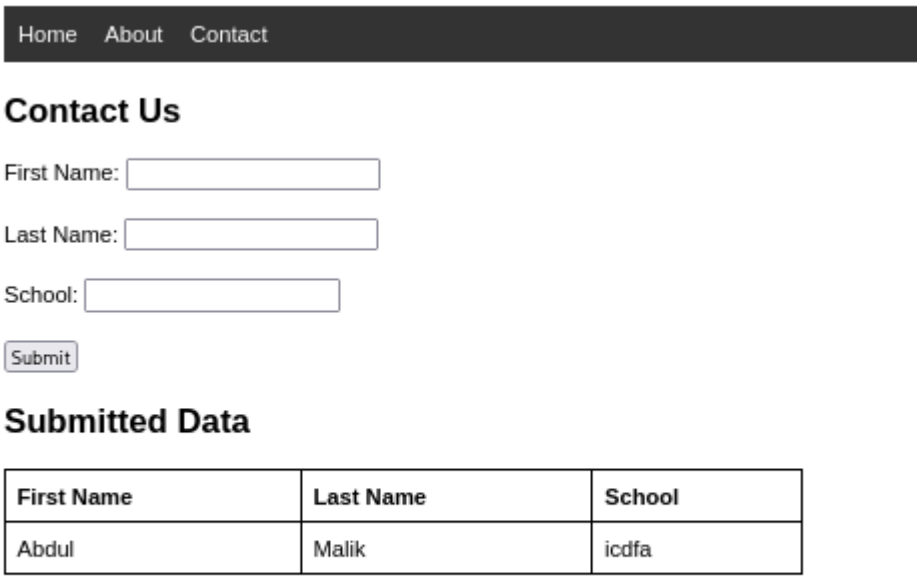
1. Identify Front-End Technologies: Research and create a list of common front-end technologies used in web development, including:

- **HTML (Hyper Text Markup Language):**
  - Provides the structure of web pages.
  - Defines elements like headings, paragraphs, lists, links, forms, and images.
- **CSS (Cascading Style Sheets):**
  - Styles and formats HTML elements.
  - Controls layout, colours, fonts, spacing, and responsive design.

- **JavaScript:**
  - Adds interactivity to web pages.
  - Enables dynamic content updates, form validation, animations, and event handling.
- **Front-End Frameworks:**
  - Tools and libraries that simplify front-end development.
  - Examples:
    - **React:** Component-based library for building interactive UIs.
    - **Angular:** Full-featured framework for building scalable web applications.
    - **Vue.js:** Lightweight framework for reactive and component-driven UI development

2. Create a Simple Front-End Application:

- Build a basic web application using HTML, CSS, and JavaScript that includes:
    - A header with a navigation menu.
    - A form for user input (e.g., contact form).
    - Interactive elements (e.g., a button that changes the text when clicked).



3. Reflection: Discuss how front-end technologies interact to create a cohesive user experience. What potential security issues arise from poorly implemented front-end code

(e.g., XSS, CSRF)?

- **HTML, CSS, and JavaScript** work together to build a good user experience.

    HTML: gives structure (forms, menus).

    - CSS: makes it look nice.
    - JavaScript: makes it interactive (buttons, validation).
- **Security issues if not done well:**
    - **XSS:** attackers inject scripts if inputs are not cleaned.
    - **CSRF:** attackers trick users into doing actions without knowing.
    - **Sensitive data leak:** showing raw data or errors.

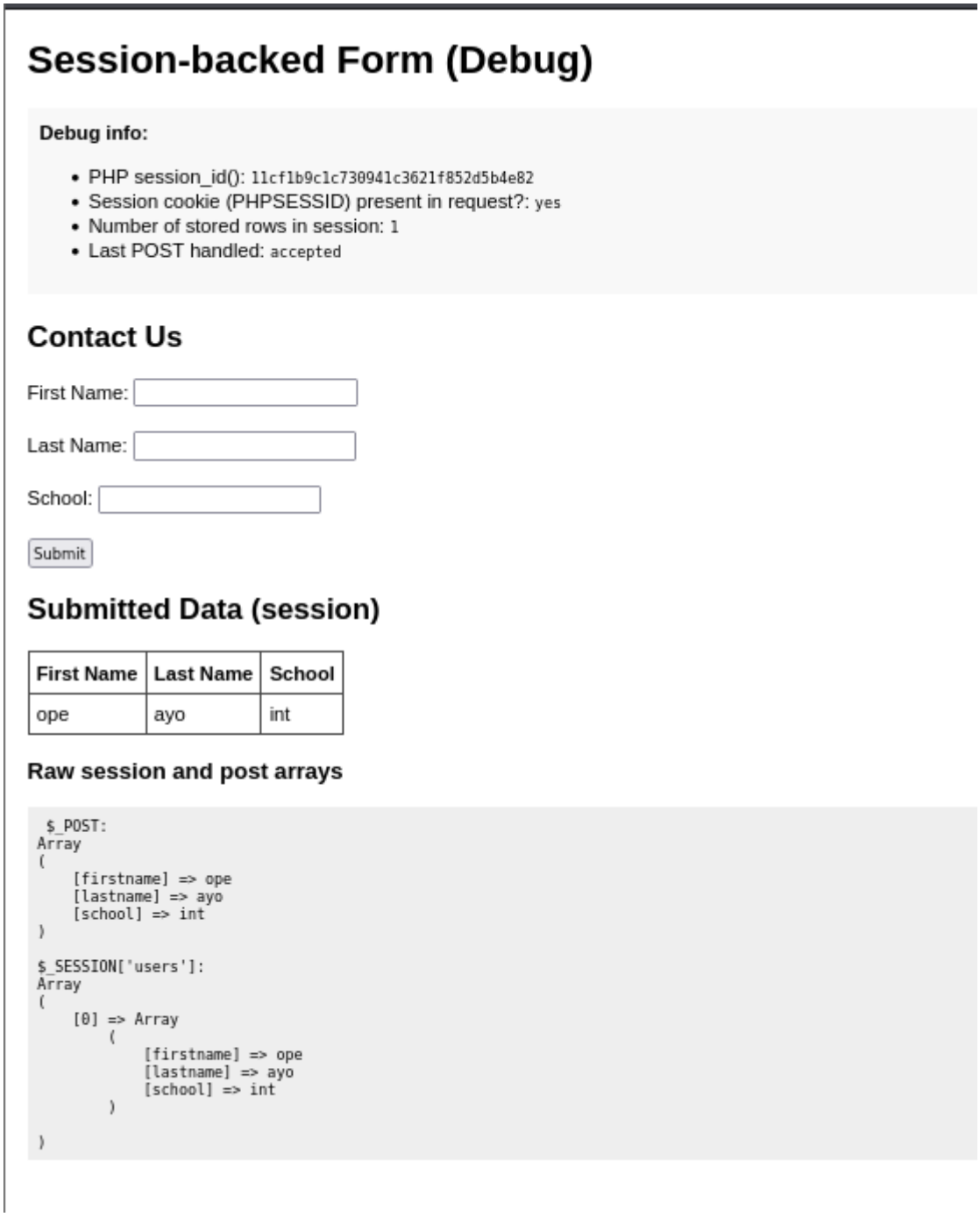
    - **Weak validation:** only checking on front-end can be bypassed.

Exercise 2: Understanding Back-End Development Components

1. Identify Back-End Technologies: Create a list of common back-end technologies, including:

- **Server-Side Programming Languages:**
    - **PHP:** Widely used for web development and dynamic page generation.
    - **Python:** Often used with frameworks like Django and Flask for web applications.
    - **Java:** Used for enterprise-level applications and web services.
    - **Node.js:** JavaScript runtime for building scalable server-side applications.
- **Databases:**
    - **MySQL:** Relational database for structured data.
    - **MongoDB:** NoSQL database for flexible, document-based storage.
    - **PostgreSQL:** Advanced relational database with strong ACID compliance.
- **Web Frameworks:**
    - **Express (Node.js):** Minimalist framework for building APIs and web applications.
    - **Django (Python):** High-level framework for rapid development and security.
    - **Laravel (PHP):** Modern PHP framework for building robust web applications.

2. Create a Simple Back-End Application: Using a chosen back-end technology, set up a basic server that responds to HTTP requests. For example, using Node.js:



Session-backed Form (Debug)

Debug info:

- PHP session_id(): 11cf1b9c1c730941c3621f852d5b4e82
- Session cookie (PHPSESSID) present in request?: yes
- Number of stored rows in session: 1
- Last POST handled: accepted

Contact Us

First Name:

Last Name:

School:

Submit

Submitted Data (session)

First Name	Last Name	School
ope	ayo	int

Raw session and post arrays

```
 $_POST:
Array
(
    [firstname] => ope
    [lastname] => ayo
    [school] => int
)

$_SESSION['users']:
Array
(
    [0] => Array
        (
            [firstname] => ope
            [lastname] => ayo
            [school] => int
        )

)
```

Reflection: Role of Back-End Components and Vulnerabilities

The back-end of a web application handles critical tasks such as processing user input, storing data, authenticating users, and communicating with databases. It ensures that the application works behind the scenes and delivers the correct information to the front-end.

However, vulnerabilities in the back-end can severely compromise overall application security:

- **SQL Injection** – If user input is not properly validated, attackers can manipulate database queries, leading to unauthorized data access, data loss, or even full control of the database.
- **Insecure APIs** – Poorly protected APIs may expose sensitive data or allow attackers to bypass authentication and authorization checks.
- **Weak Session Management** – If sessions are not properly secured, attackers can hijack user accounts.
- **Poor Input Validation** – Malicious inputs can lead to XSS, code execution, or logic bypasses.

Exercise 3: Identifying Security Threats

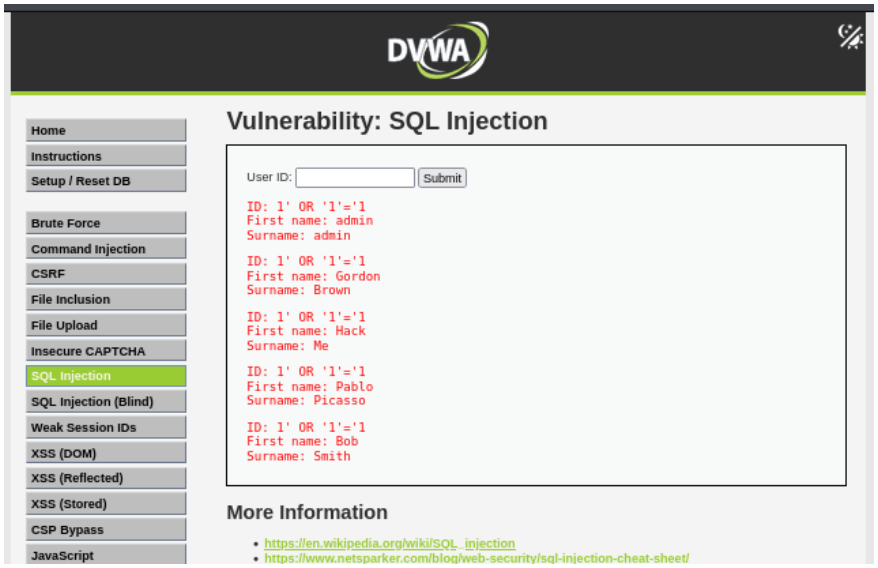
1. Research Common Security Threats:

- Investigate the OWASP Top Ten list and identify at least five common security threats in web development. Focus on:
    - **SQL Injection (SQLi)**
      Attackers manipulate database queries through unsensitized input. Can lead to stolen, modified, or deleted data.
    - **Cross-Site Scripting (XSS)**
      Malicious scripts are injected into web pages and executed in the victim's browser. Can steal cookies, hijack sessions, or deface sites.
    - **Cross-Site Request Forgery (CSRF)**
      Tricks a user's browser into performing unauthorized actions (like changing passwords) while logged in.
    - **Security Misconfiguration**
      Poorly configured servers, databases, or frameworks expose sensitive information. Examples: default credentials, verbose error messages.
    - **Insecure Deserialization**
      Exploiting insecure handling of serialized objects to execute malicious code or escalate privileges.

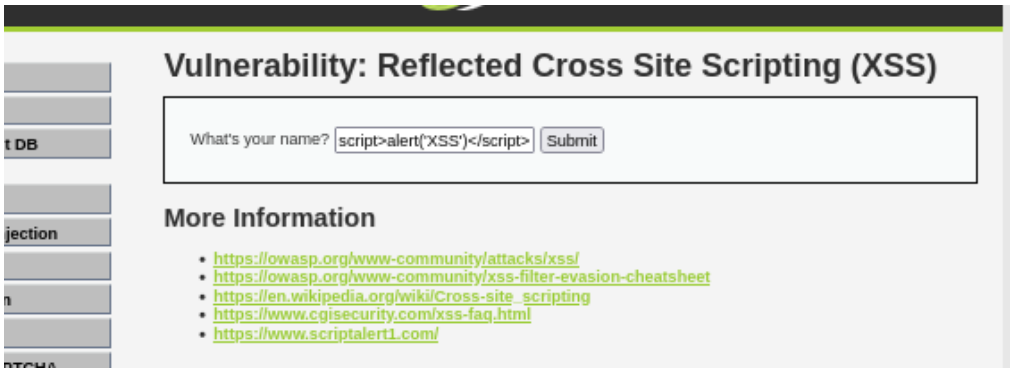
2. Analyze a Sample Application:

- Use a vulnerable web application (e.g., DVWA or OWASP Juice Shop) to test for identified vulnerabilities.
- Perform the following tasks:

- Attempt an SQL injection attack using a login form.

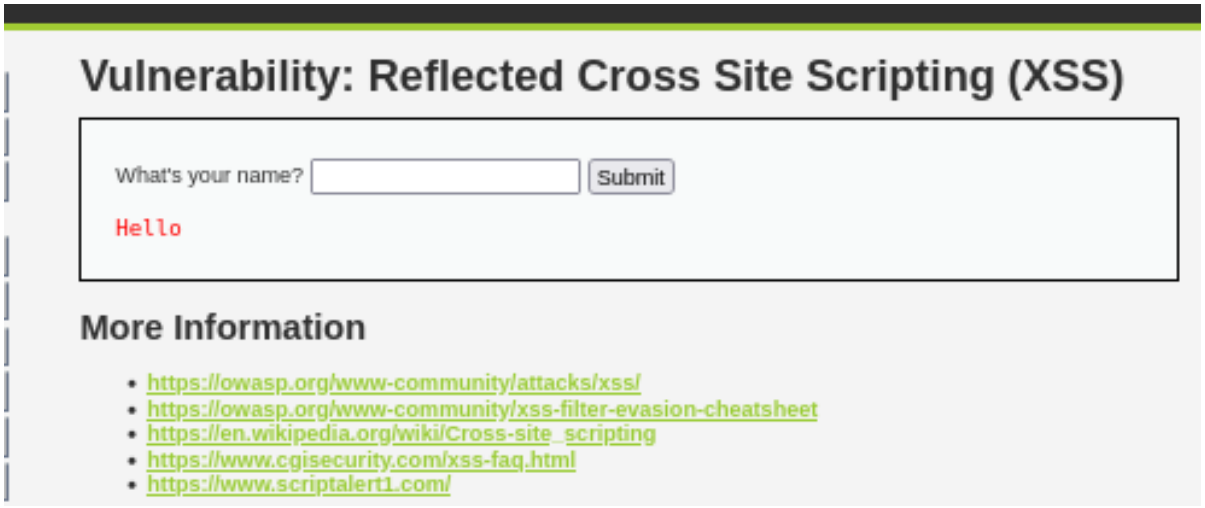


- Test for XSS by injecting scripts into input fields.
  - Script inputed

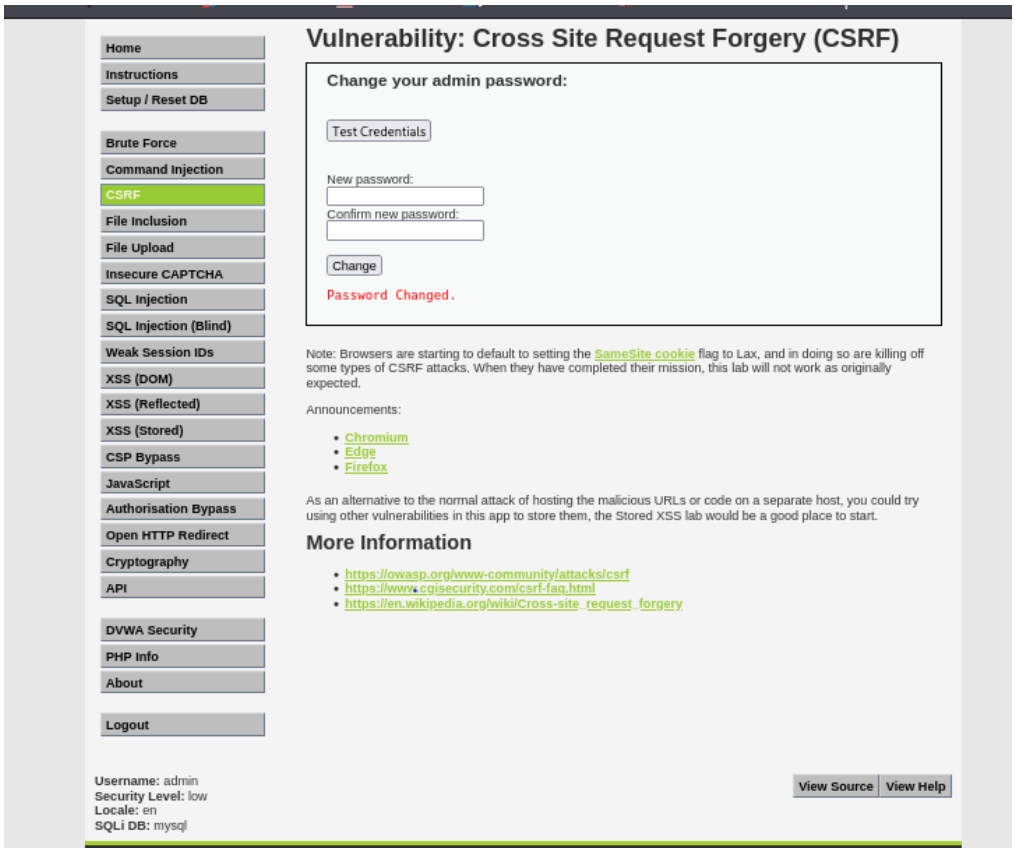


  - Alert Generated

- Output/Result



- Check for CSRF by submitting a request to change user data without authentication.



3. Reflection: Discuss the impact of these security threats on user data and application integrity. What measures can developers implement to mitigate these risks?

Reflection: Impact of Security Threats

- Attackers could bypass authentication, gain unauthorized access, and potentially manipulate or steal sensitive data
- By injecting scripts into input fields, attackers could execute malicious code in other users' browsers.
- Changing user input without authentication shows that attackers can trick users' browsers into performing unintended actions

Mitigation measures developers can implement:

- **Prevent SQL Injection:**
- **Prevent XSS:**
- **Prevent CSRF**
- **General security practices:**