|  |  |
| --- | --- |
| Ex.No.14 | **SQL Injection – Damn Vulnerable Web Application** |

|  |
| --- |
| **AIM:** |

To demonstrate SQL injection attack and Cross Site Scripting using Damn Vulnerable Web Application (DVWA).

|  |
| --- |
| **THEORY:** |

**About DVWA**

**Damn Vulnerable Web Application (DVWA)** is an open-source web application specifically designed for educational purposes in the field of web application security. It provides a platform for security professionals, developers, and enthusiasts to practice and learn about various vulnerabilities, such as SQL injection, cross-site scripting (XSS), and command injection, in a controlled environment.

**SQL injection Attack**

**SQL Injection (SQLi)** is a type of security vulnerability that allows attackers to manipulate SQL queries by injecting malicious code through user input. This can lead to unauthorized access to sensitive data, modification or deletion of database records, and execution of administrative operations on the database.

**Ways to Prevent SQL Injection**

1. **Use Parameterized Queries**:
   * Employ prepared statements to treat user input as data, not executable code.
   * Implement stored procedures to prevent direct execution of SQL commands by attackers.
2. **Input Validation and Sanitization**:
   * Validate and sanitize all user inputs to conform to expected formats. Define valid user inputs and reject abnormal queries to minimize attack vectors.
3. **Escaping Inputs**:
   * Use functions like mysqli\_real\_escape\_string() to neutralize special characters.
4. **Enforce Least-Privilege Access**:
   * **Limit User Privileges**: Grant users only the access necessary for their roles, reducing potential damage from compromised accounts.
   * **Restrict Shared Access**: Limit database access across multiple applications to minimize risk exposure.

**Cross site scripting**

**Cross-Site Scripting (XSS)** is a security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. This can lead to unauthorized actions, such as data theft, session hijacking, and the execution of arbitrary code in the context of the victim's browser. XSS attacks typically occur when user input is not properly validated or sanitized, enabling attackers to manipulate the content displayed on a website.

**Types of Cross-Site Scripting (XSS)**

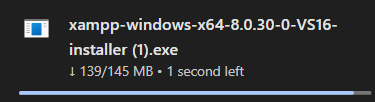
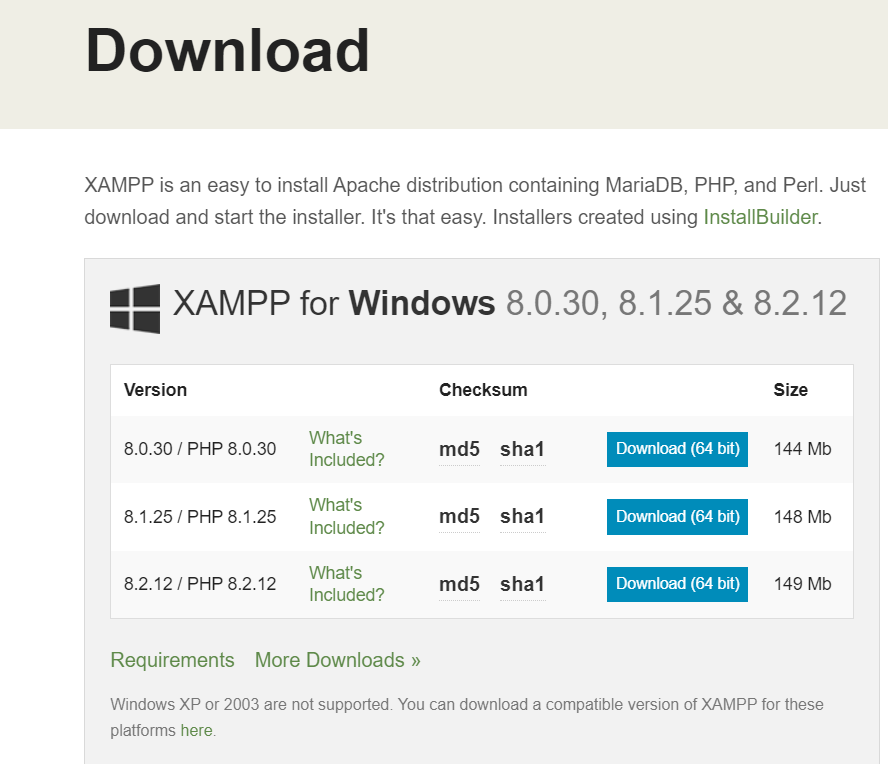
1. **Reflected XSS**:
   * **Description**: This occurs when an application immediately reflects user input back in the response without proper validation. It typically happens through URL parameters.
   * **Example**: An attacker crafts a URL that includes a malicious script, which is then executed when the victim clicks the link.
2. **Stored XSS (Persistent XSS)**:
   * **Description**: In this type, malicious scripts are stored on the server (e.g., in a database) and served to users when they access affected pages.
   * **Example**: A user submits a comment containing a malicious script, which is saved and later displayed to other users.
3. **DOM-Based XSS**:
   * **Description**: This type occurs when client-side scripts modify the DOM(Document Object Model) in an unsafe manner, allowing attackers to execute scripts based on manipulated data from the browser.
   * **Example**: A web page retrieves a URL parameter and directly writes it to the DOM without input sanitization or validation, allowing an attacker to execute scripts.

|  |
| --- |
| **Worksheet (Answer the following ques and paste the relevant outputs)** |

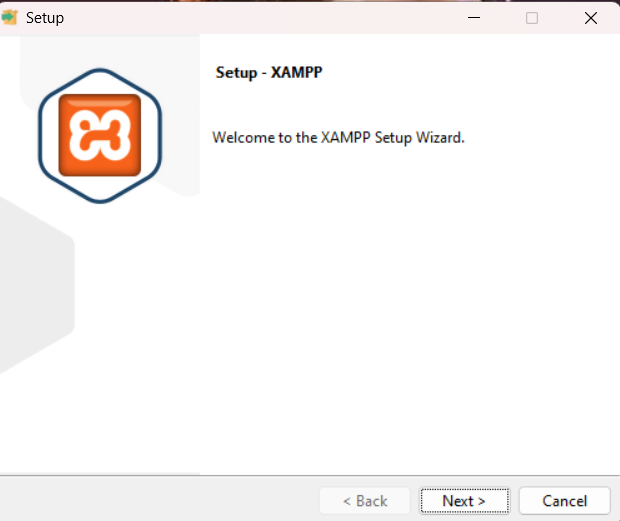
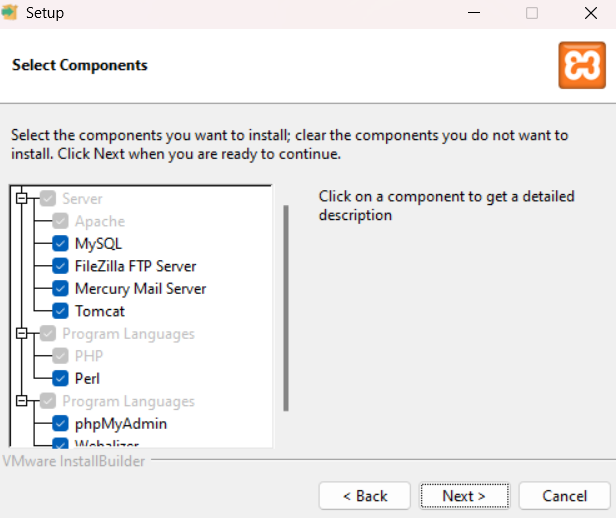
**Installation procedure for XAMPP and DVWA:**

**XAMPP:**

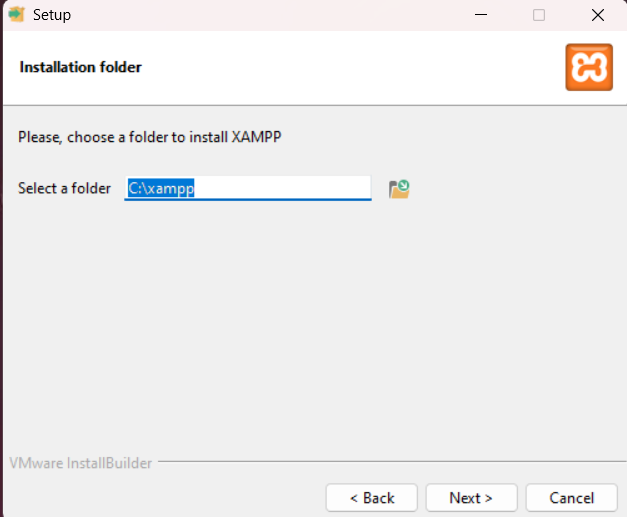
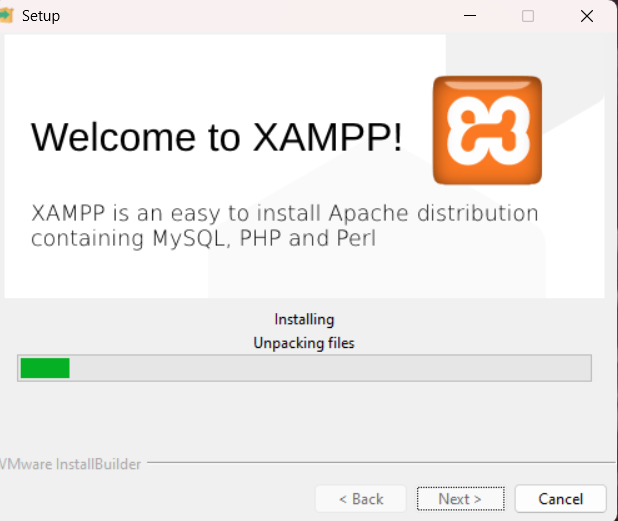
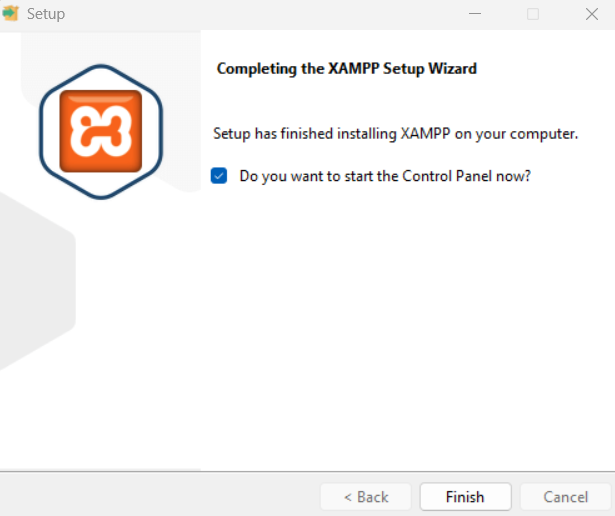
1. Install XAMPP from the official website <https://www.apachefriends.org/download.html>.



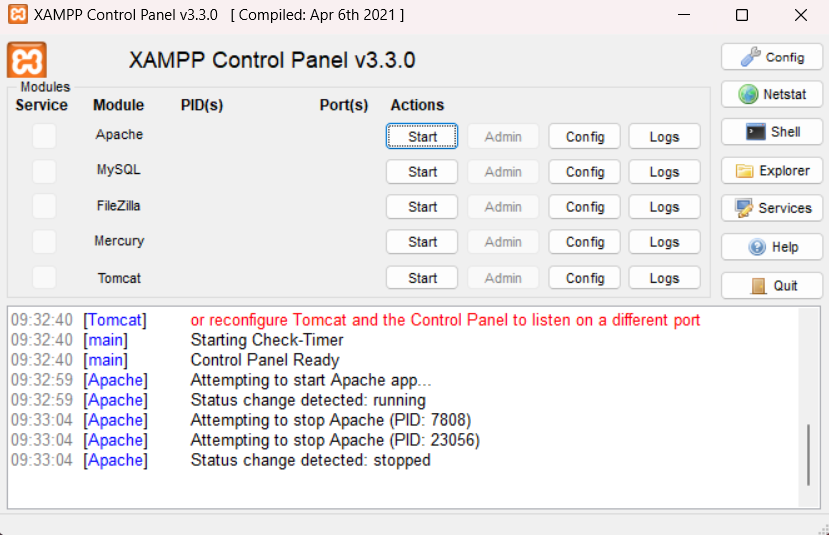
1. Run the XAMPP installer, and the wizard will open. Click ‘Next’.

1. Choose the folder for XAMPP and click "Next" until the "Install" button is enabled.

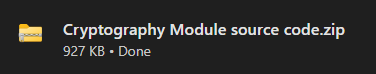
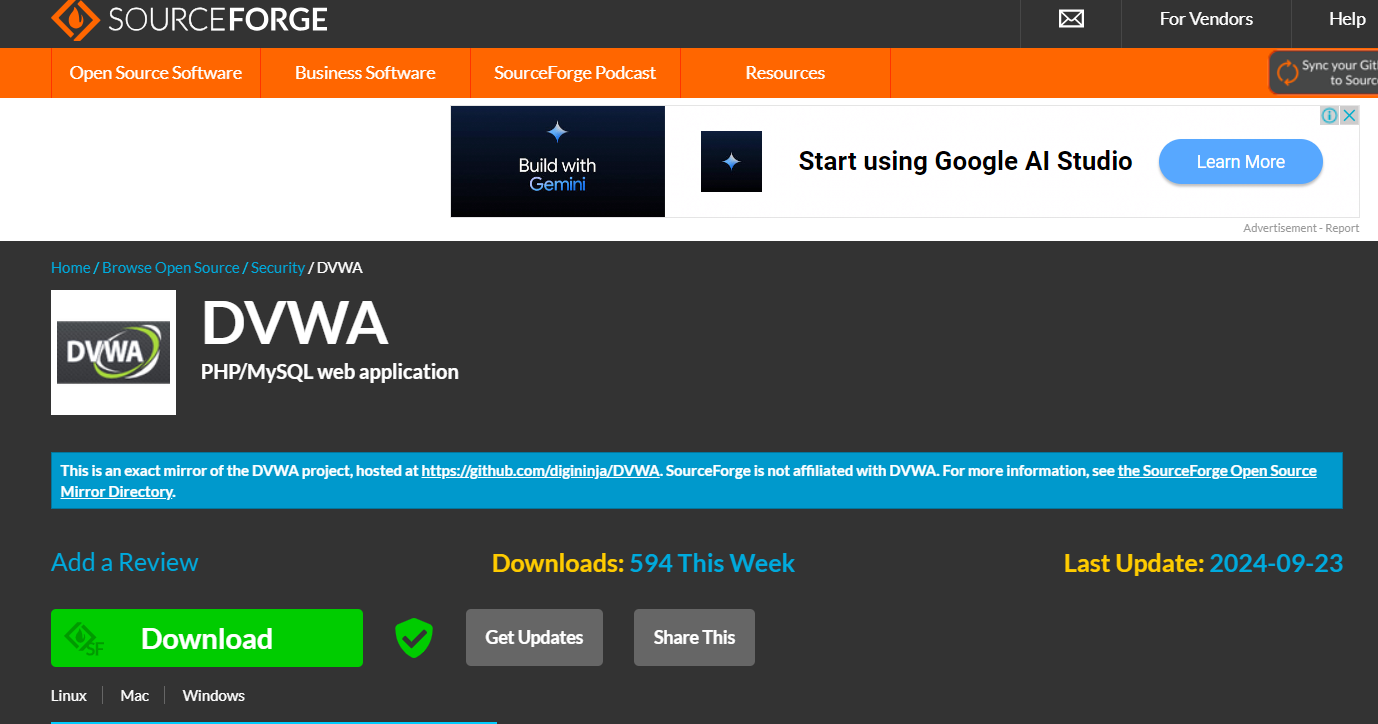
 ****

1. **XAMPP** has been successfully installed.

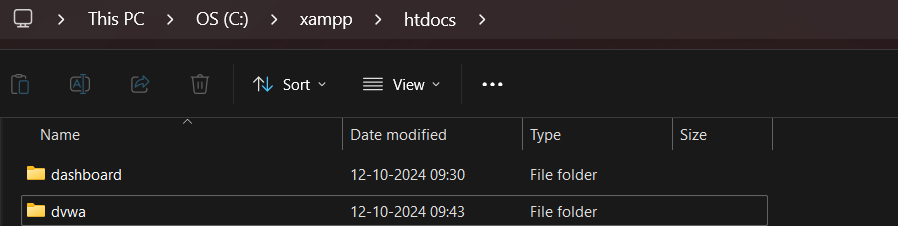
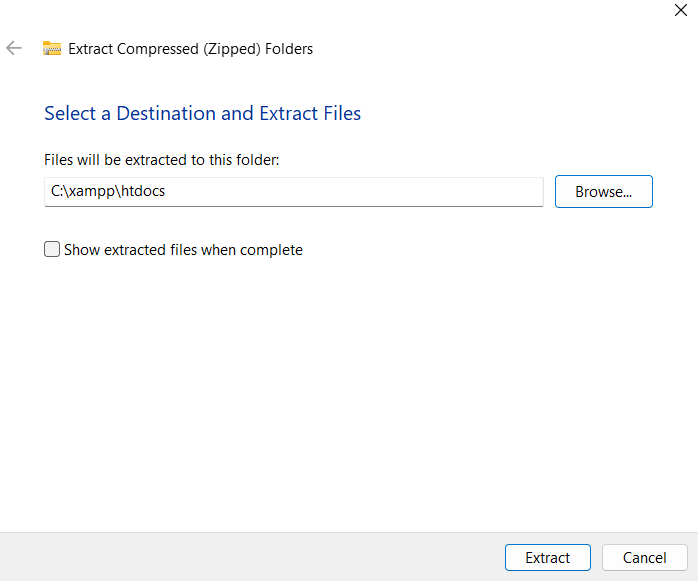
****

**Damn Vulnerable Web Application (DVWA):**

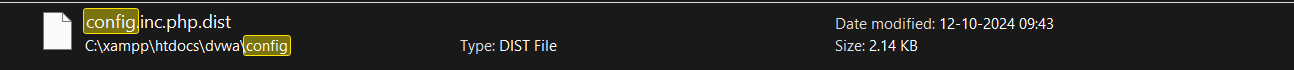
1. Download DVWA from the link <https://sourceforge.net/projects/dvwa.mirror/> It will be downloaded as a zip file.

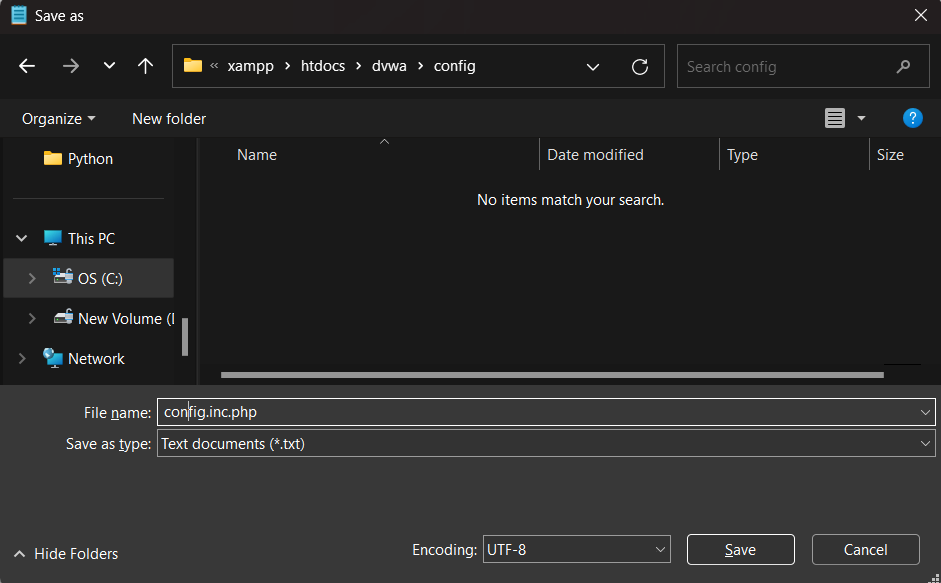


1. Now extract the downloaded ZIP folder to the C:\xampp\htdocs folder and rename the folder as dvwa.



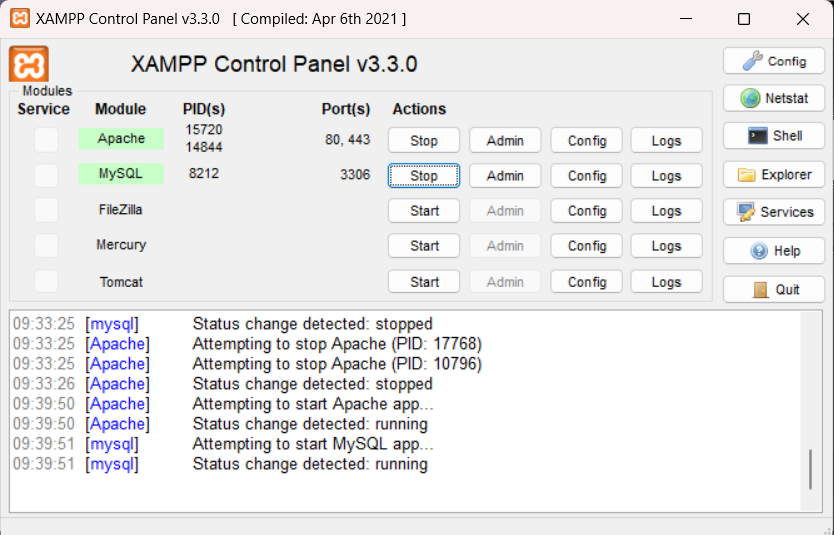
1. Now go to C:\xampp\htdocs\dvwa\config. You might find a file named config.inc.php.dist. Copy the contents of this file, create a new file, paste the content, and name it config.inc.php



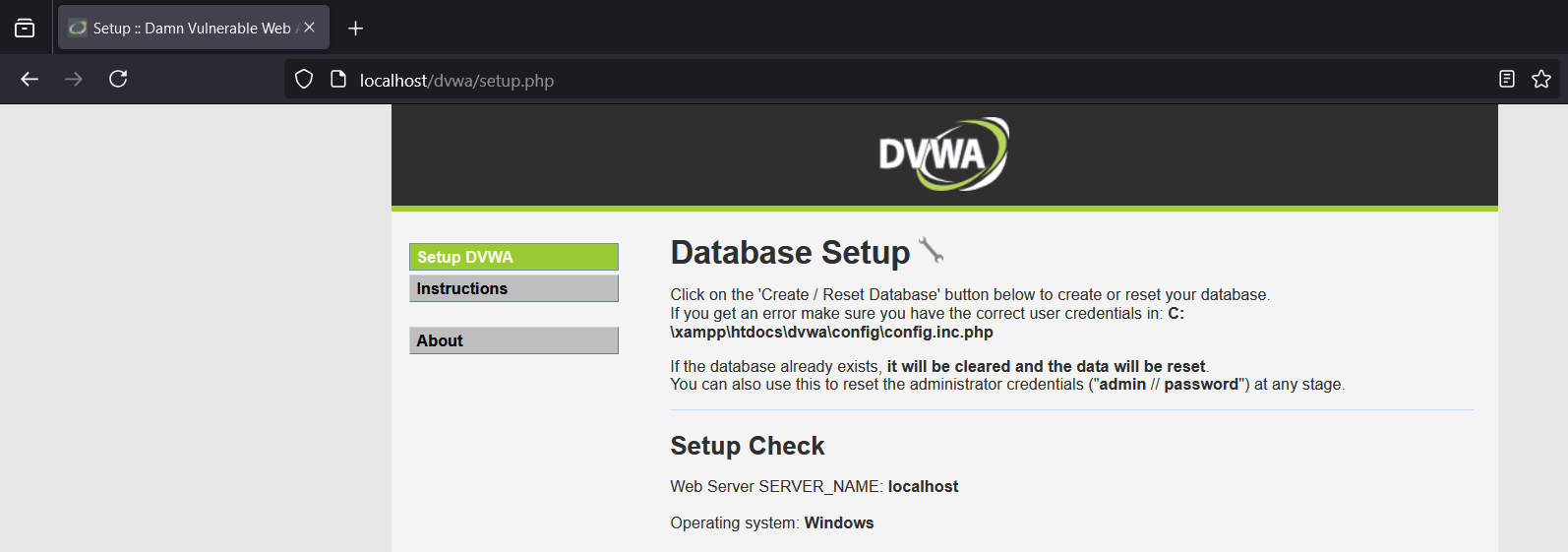




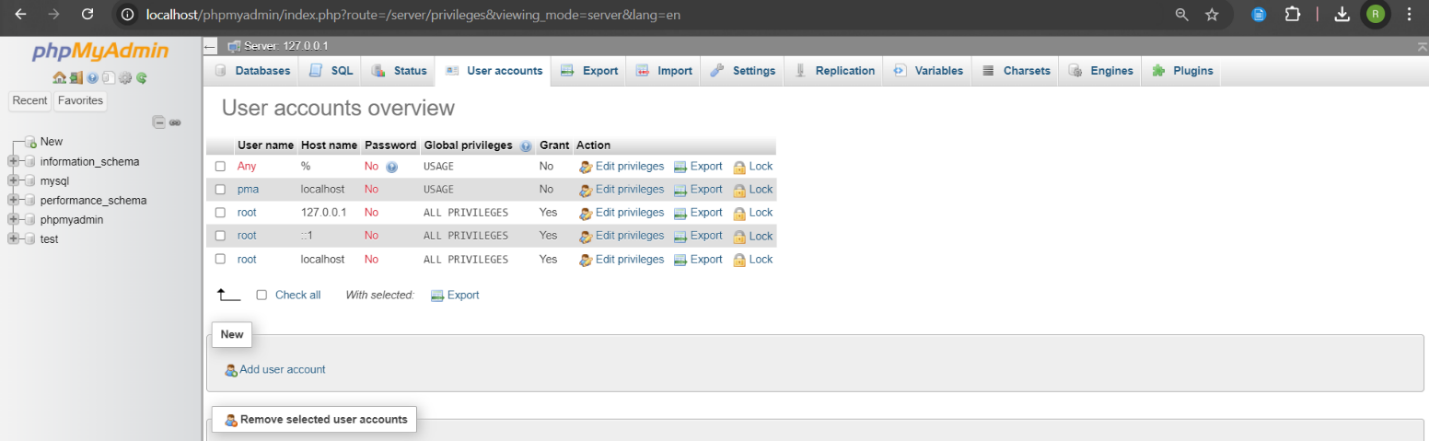
1. Start the Apache and MySQL services in the XAMPP Control Panel.

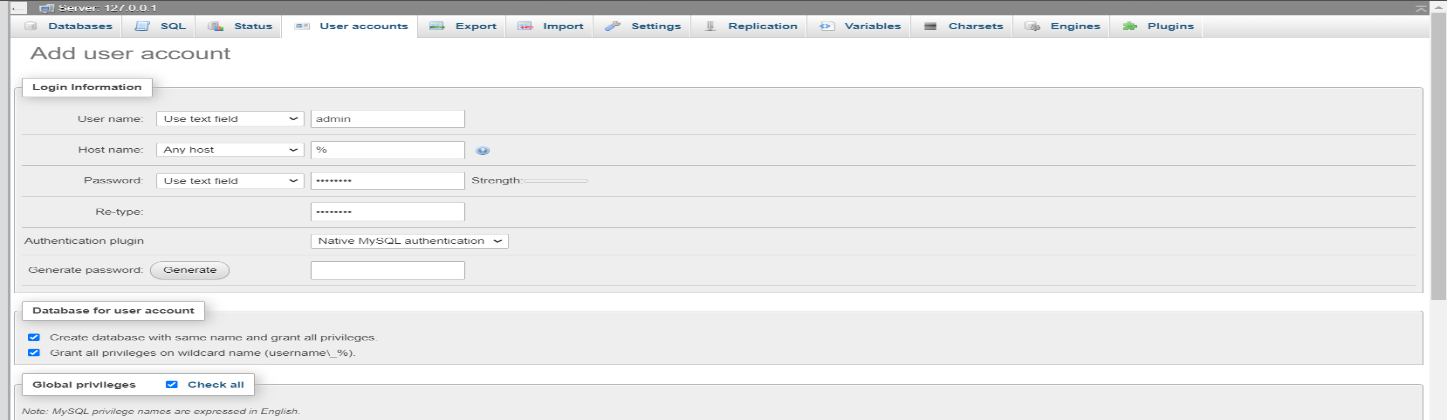
****

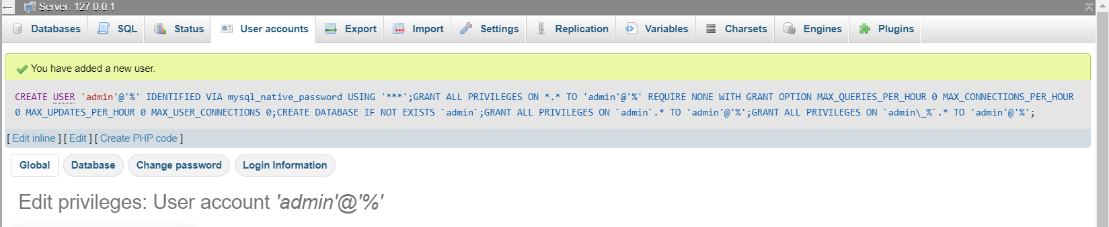
1. Go to firefox and type localhost/dwva/setup.php and verify the successful installation of dwva.



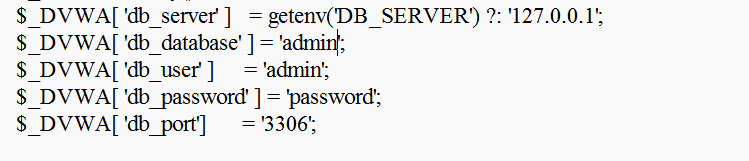
1. To create a database, we need to add a user in MySQL. Go to browser > type localhost/phpMyAdmin/index.php>User Accounts > Create a user with the username "admin" and the password "password," and grant all privileges to this account.



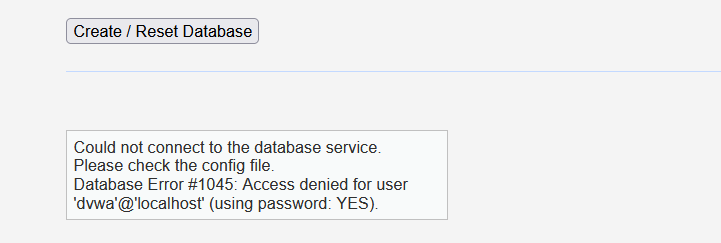


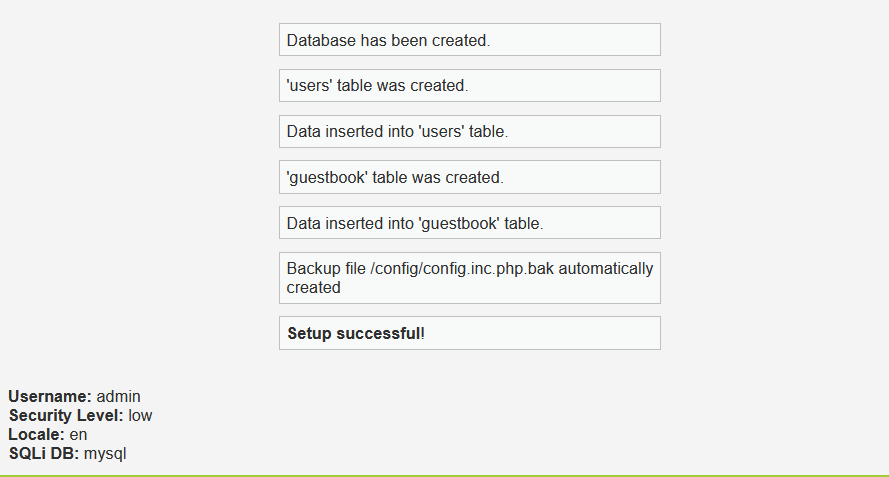


1. Edit the config.inc.php file to update the database name, username, and password.



1. Open Firefox and navigate to localhost/dvwa/setup.php. Click on "Create/Reset Database" at the bottom to create or reset the database.



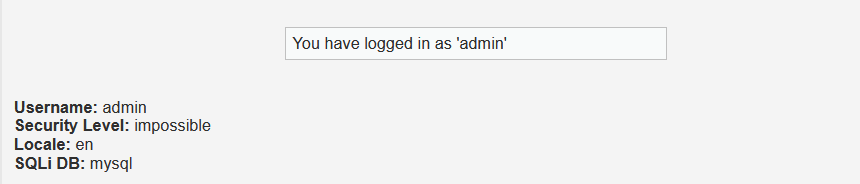
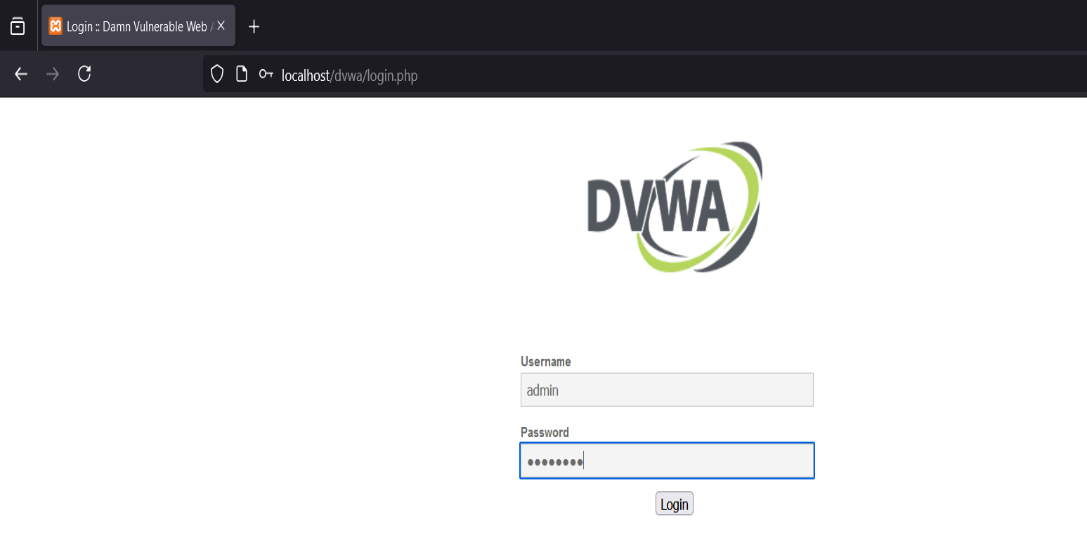


Database has been successfully created.

1. Now log in to DVWA by entering localhost/dvwa/login.php in the address field of Firefox.

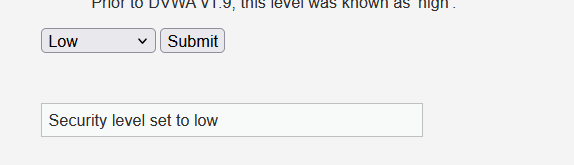
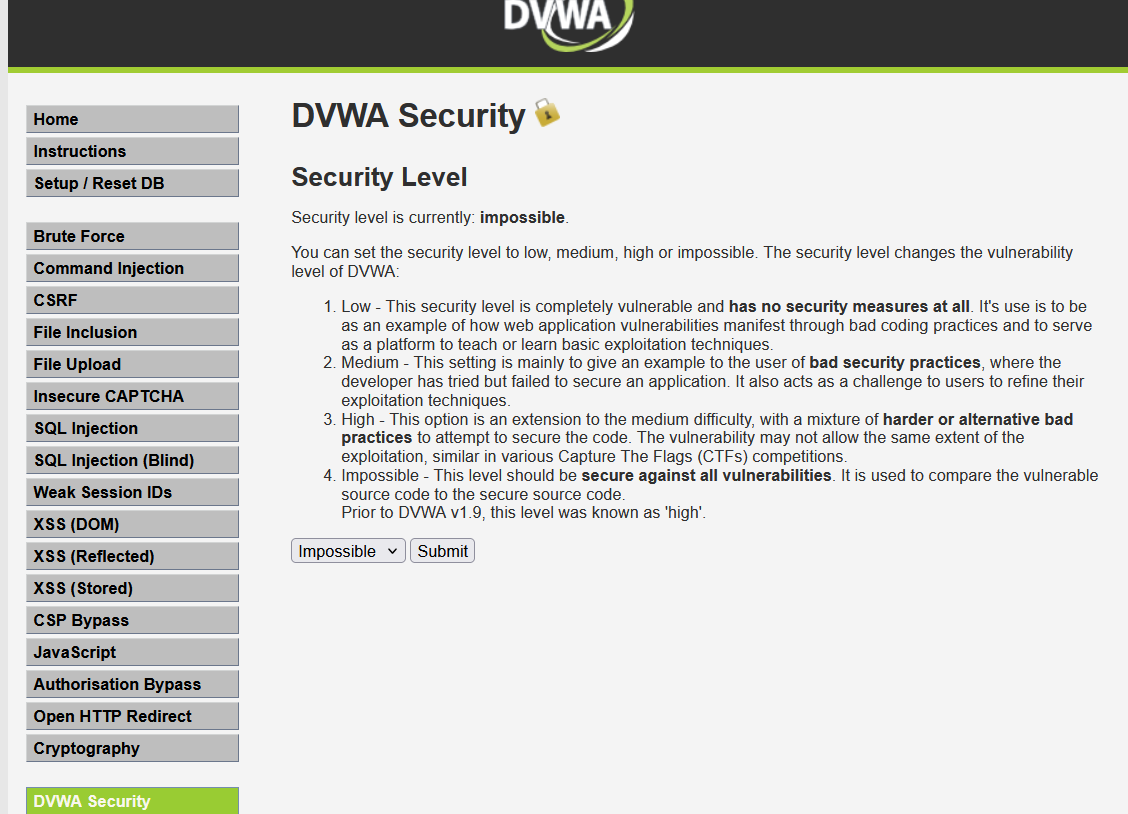
Credentials for login page are:

* + Username : admin Password : password

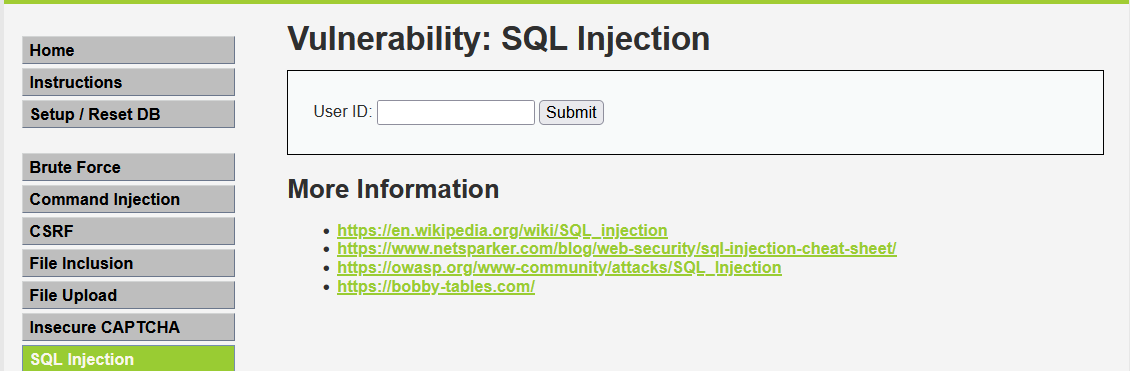


**SQL INJECTION ATTACK:**

* After logging in, go to “DVWA security” section. There is a drop down box on that page.
* You can set the security level of this web application accordingly. Set it “low” which is the least secure mode.



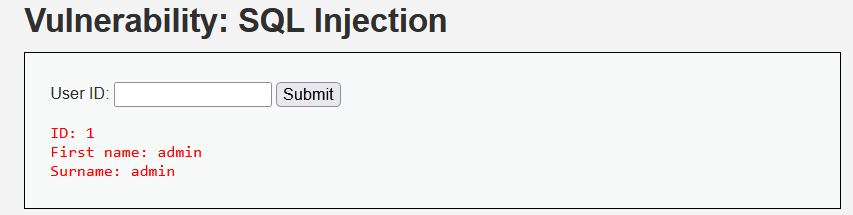
* Now move to SQL injection section, by clicking “sql injection” button on page. And try SQL injection there.



**Security level is in low:**

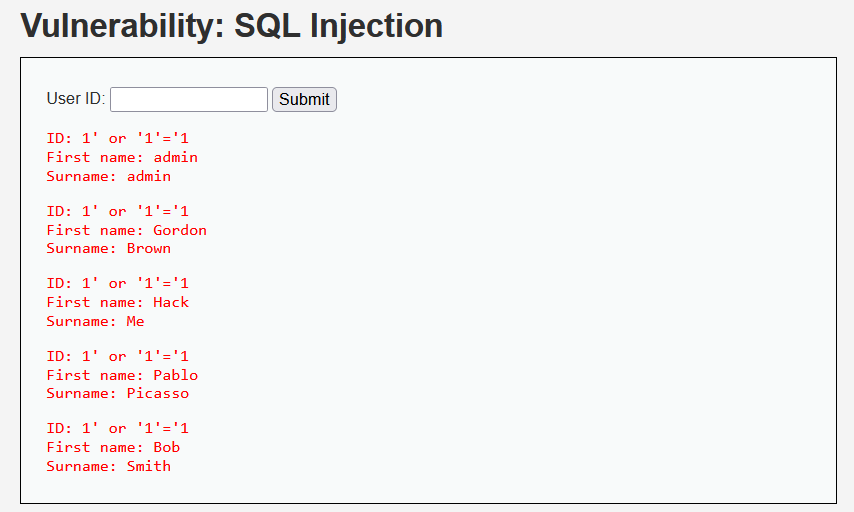
1. Check expected results:

SELECT first\_name, last\_name FROM users WHERE user\_id = 1



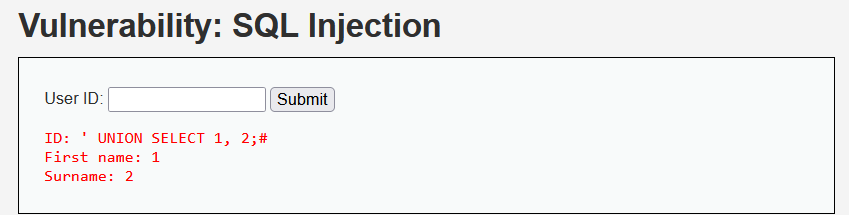
1. Check the results of an OR True statement

SELECT first\_name, last\_name FROM users WHERE user\_id = 1' or '1'='1



1. Find the number of columns in table:

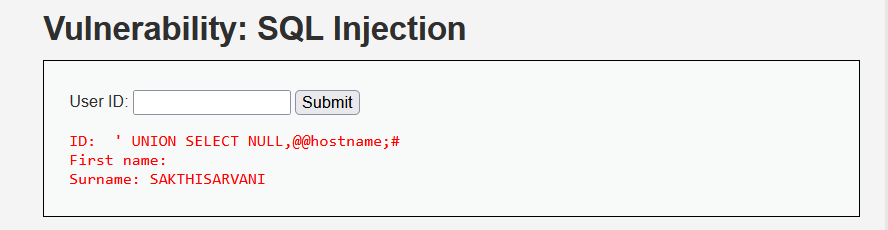
SELECT first\_name, last\_name FROM users WHERE user\_id = ' UNION SELECT 1, 2;#



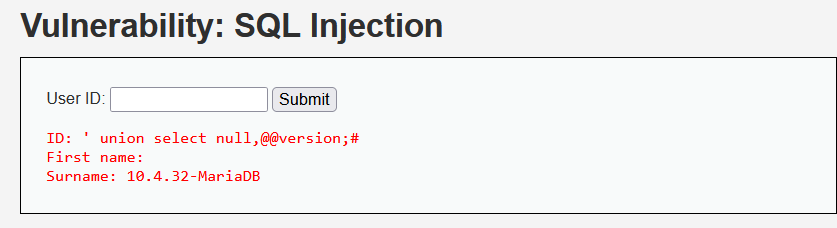
This shows that there are atleast 2 columns

1. Find Hostname:

SELECT first\_name, last\_name FROM users WHERE user\_id = ' union select null,@@hostname;#

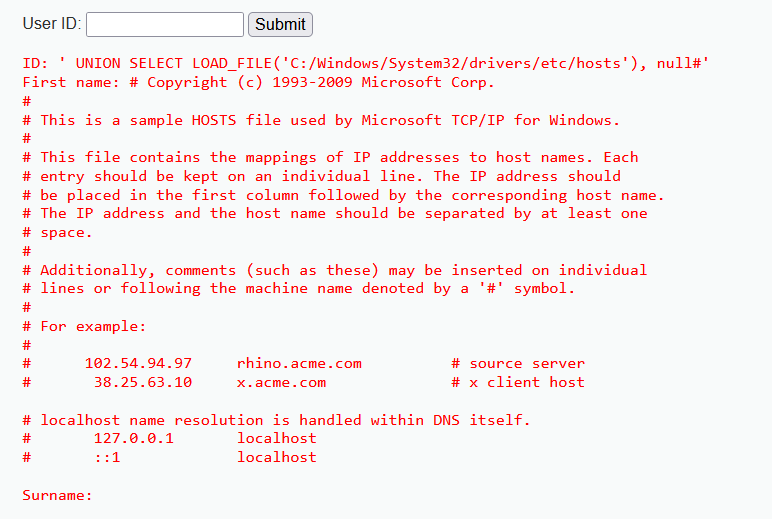


1. Try to do SQL Injection on DVWA with security level set to “Medium” and find out mysql version. 1 union select null,@@version#



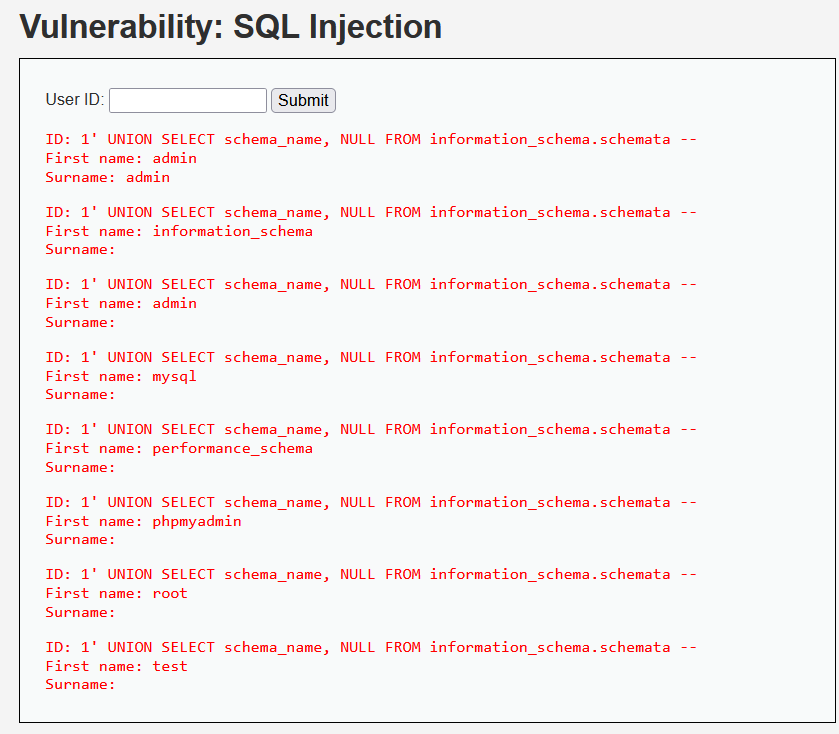
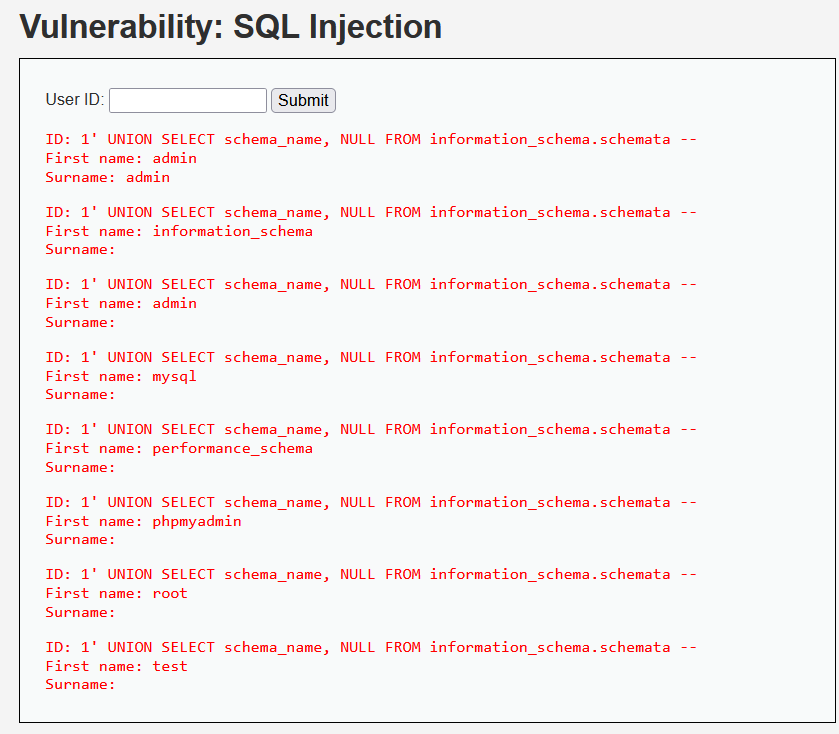
1. Display File:

SELECT first\_name, last\_name FROM users WHERE user\_id = ' UNION SELECT LOAD\_FILE('C:/Windows/System32/drivers/etc/hosts'), null#’

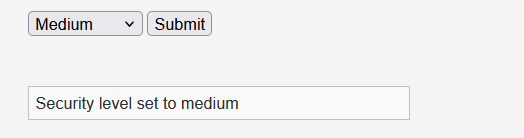


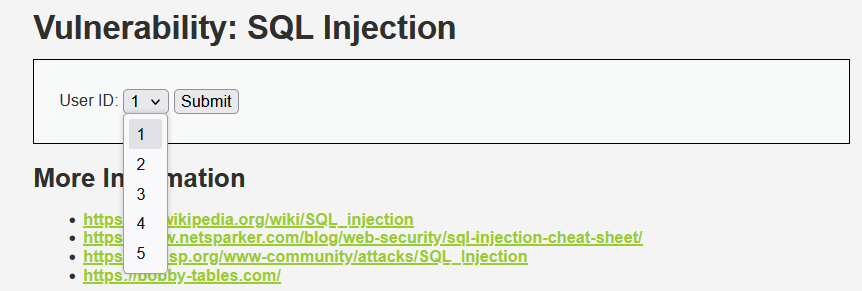
1. Try to do SQL Injection on DVWA and find out all schema name from database.

1' UNION SELECT schema\_name, NULL FROM information\_schema.schemata –



**Security Level is in Medium:**





6. Try to do SQL Injection on DVWA with security level set to “Medium” and find out all schema name from database.

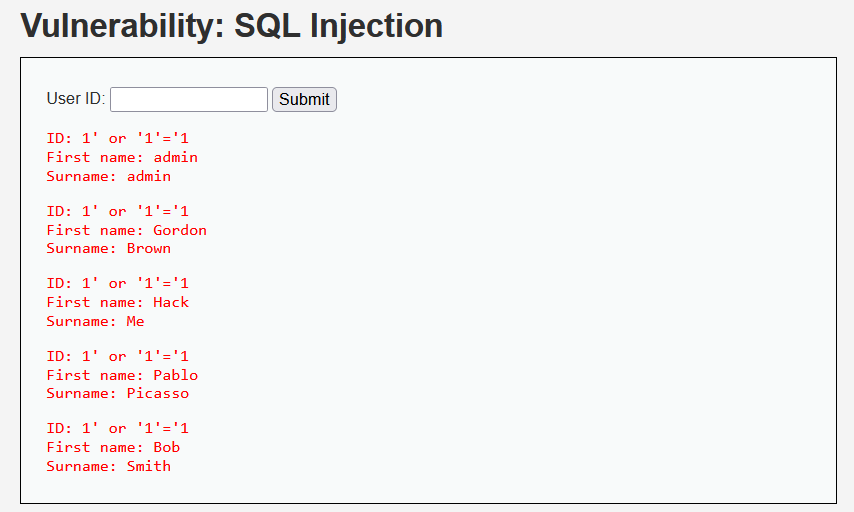
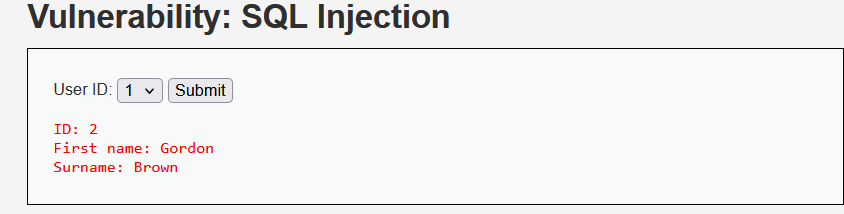
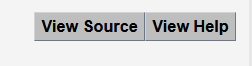
1 or 1=1 union select null, table\_name from information\_schema.tables#

If the security level is set to "Medium," we cannot type input; we can only choose an ID from a dropdown. Therefore, we cannot execute the command

7.Try to do SQL Injection on DVWA with security level set to “Medium” and find out mysql version.

1 union select null,@@version#

If the security level is set to "Medium," we cannot type input; we can only choose an ID from a dropdown. Therefore, we cannot execute the command



Security Level : Medium Low

1. Analyze the source code for security on “low” and “medium” level. Comment on the ways by which SQL injection is prevented.

**Low Security Level**

* **Vulnerability:**
  + The code directly interpolates user input ($id) into the SQL query without any sanitization, allowing for SQL injection attacks.

$query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

* **Prevention Measures:**
  + There are no mechanisms in place to prevent SQL injection.
* **Comments:**
  + Input sanitization is absent and Vulnerable to various SQL injection techniques.

**Medium Security Level**

* **Vulnerability:**
  + User input is sanitized using mysqli\_real\_escape\_string(), which helps mitigate some SQL injection attempts.

 $id = mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $id);

* + However, the method still allows for potential vulnerabilities if input is manipulated cleverly.
* **Prevention Measures:**
  + Input sanitization with mysqli\_real\_escape\_string(), which provides limited protection against simple injections.
* **Comments:**
  + While this level provides some protection, it is not foolproof, as attackers can bypass input sanitization. For instance, by modifying the dropdown selection or input method, they may execute a crafted payload.

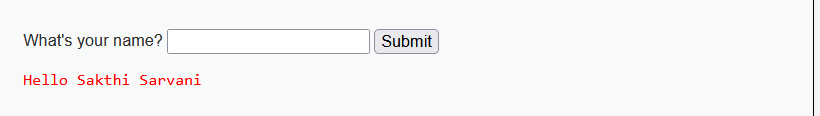
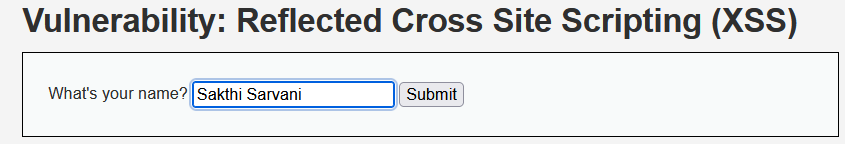
**Medium Security Level implements basic input sanitization, which can handle some sophisticated SQL injection attacks but still remains vulnerable to advanced techniques.**

**CROSS SIDE SCRIPTING:**

1. **Reflected XSS:**

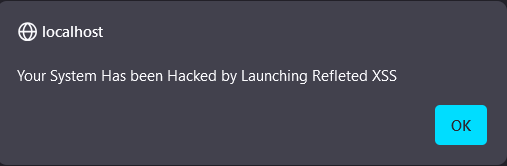
***Security Level Low:***

Any input typed in the text box will get executed.

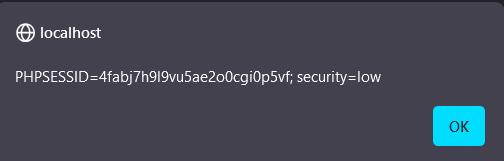
****

Try entering the script as a value in the field:

**<script>alert("Your system has been hacked by Launching Relected XSS")</script>**

****

It seems that the data entered in the text field is not validated, and the HTML code gets executed. This can lead to a breach of data security. For instance, we would be able to access the session ID using the following command: **<script>alert(document.cookie)</script>.** With the session ID, the attacker can impersonate the legitimate user."

****

***Security Level Medium:***

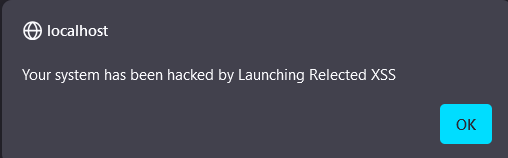
By changing the security level to medium, the user data gets validated, and the alert prompt is not shown.

**<script>alert("Your system has been hacked by Launching Relected XSS")</script>**

****

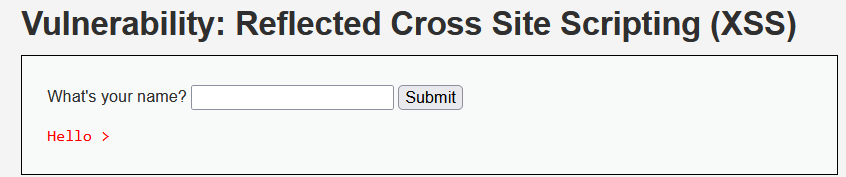
However, we can also execute scripts at medium level by manipulating the script tag**.**

**<scr<script>ipt>alert("Your system has been hacked by Launching Relected XSS")</script>**

****

***Security Level High:***

In high level security we can’t execute script.

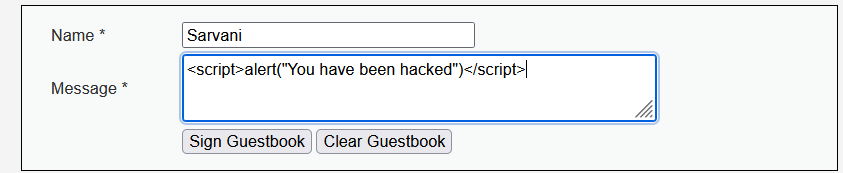
****

1. **Stored XSS:**

In Stored XSS, the data is saved in the server database, and if the security level is configured to low, the data remains unvalidated. Consequently, the script provided as a message executes every time the webpage is reloaded, continuing until the database is reset.

***Security Level Low:***

In low level we can execute the script in the message textarea.

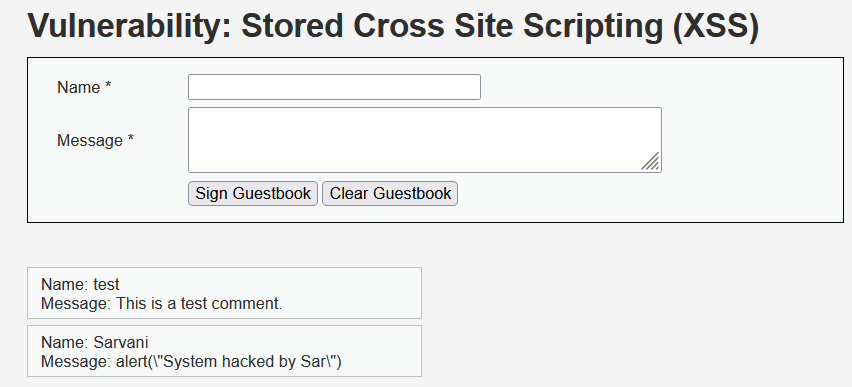


The script is executed.

***Security Level Medium:***

In medium security level, the data undergoes validation, preventing the execution of the script while it is still stored in the server database.

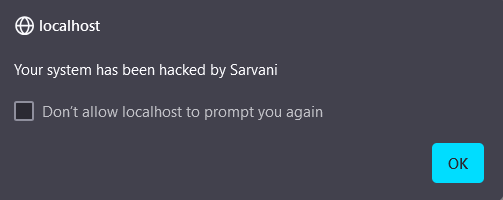
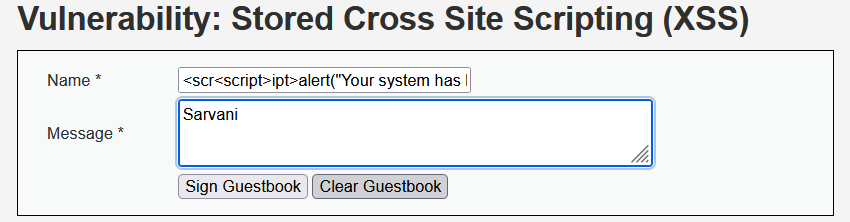
**<script>alert("System hacked by Sar")</script>**

****

However, at the medium security level, if we input a script in the name field, it will get executed.To do this, right-click and select 'Inspect (Q)' to identify the text area input. Then, increase the message length by editing it.

**<scr<script>ipt>alert("Your system has been hacked by Sarvani")</script>**

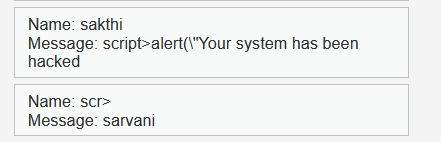
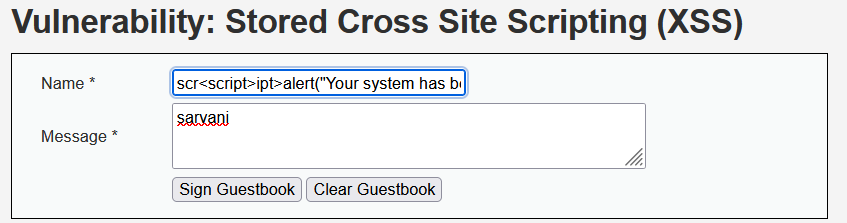
****

****

Therefore, at the medium security level, we can also inject script code by manipulating the name field, allowing the script to execute.

***Security Level High:***

In high security level, script execution is restricted. While we can manipulate the length, the script will always be stored as messages without executing

****

1. **DOM-Based XSS:**

**I**n DOM-based XSS, by modifying the client-side script, we can introduce vulnerabilities that allow attackers to manipulate the Document Object Model (DOM) and execute malicious code.

***Security Level Low and medium:***

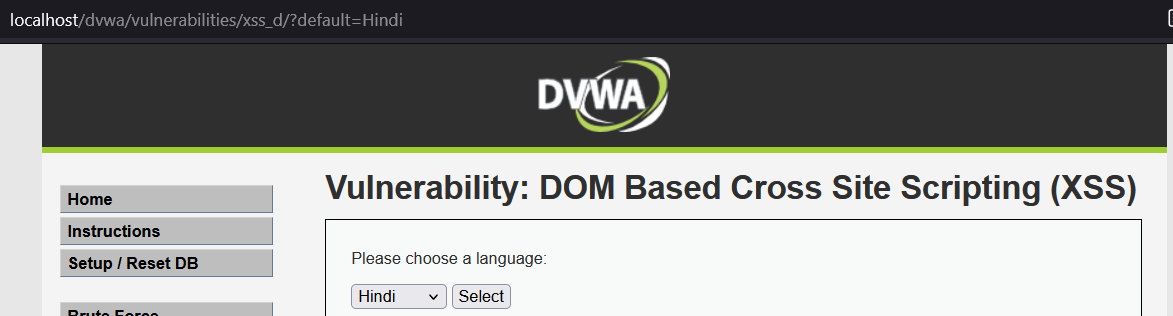
By modifying the URL content, we can adjust the DOM, allowing for dynamic changes to the webpage's elements and functionality.

The default language can be changed to Hindi or any other language, as shown below, without having an option in the dropdown if the security level is set to low or medium.

**Default values:**

****

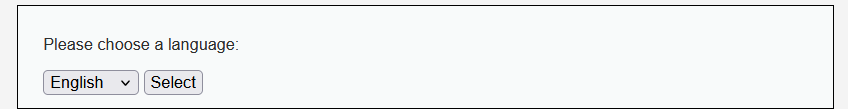
**Modifying url:**

****

***Security Level High:***

However, DOM-based XSS cannot be performed if the security level is set to high, as the client-side script cannot be modified. Even if we type in Hindi, it will dynamically change to English.

****

****

**Overview of XSS attacks in DVWA:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type of XSS** | **Low Security** | **Medium Security** | **High Security** |
| **Stored XSS** | Malicious scripts stored and executed on page load. | Data validated; scripts may still be stored, and there is a possibility to execute malicious scripts. | Strong validation prevents execution. |
| **Reflected XSS** | Scripts reflected immediately, easy exploitation. | Input validated, some scripts may execute. | Input sanitization blocks reflected scripts. |
| **DOM-based XSS** | Client-side scripts manipulated via URL values. | Some manipulation allowed, limited effectiveness. | Client-side scripts cannot be modified. |

|  |
| --- |
| **RESULT:** |

Thus, SQL injection attacks and cross-site scripting (XSS) have been demonstrated using the Damn Vulnerable Web Application (DVWA).

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Originality of the work | 30 |  |
| Completion of experiment on time | 10 |  |
| Documentation | 10 |  |
| Total | 50 |  |
| Signature of the faculty with Date |  |  |