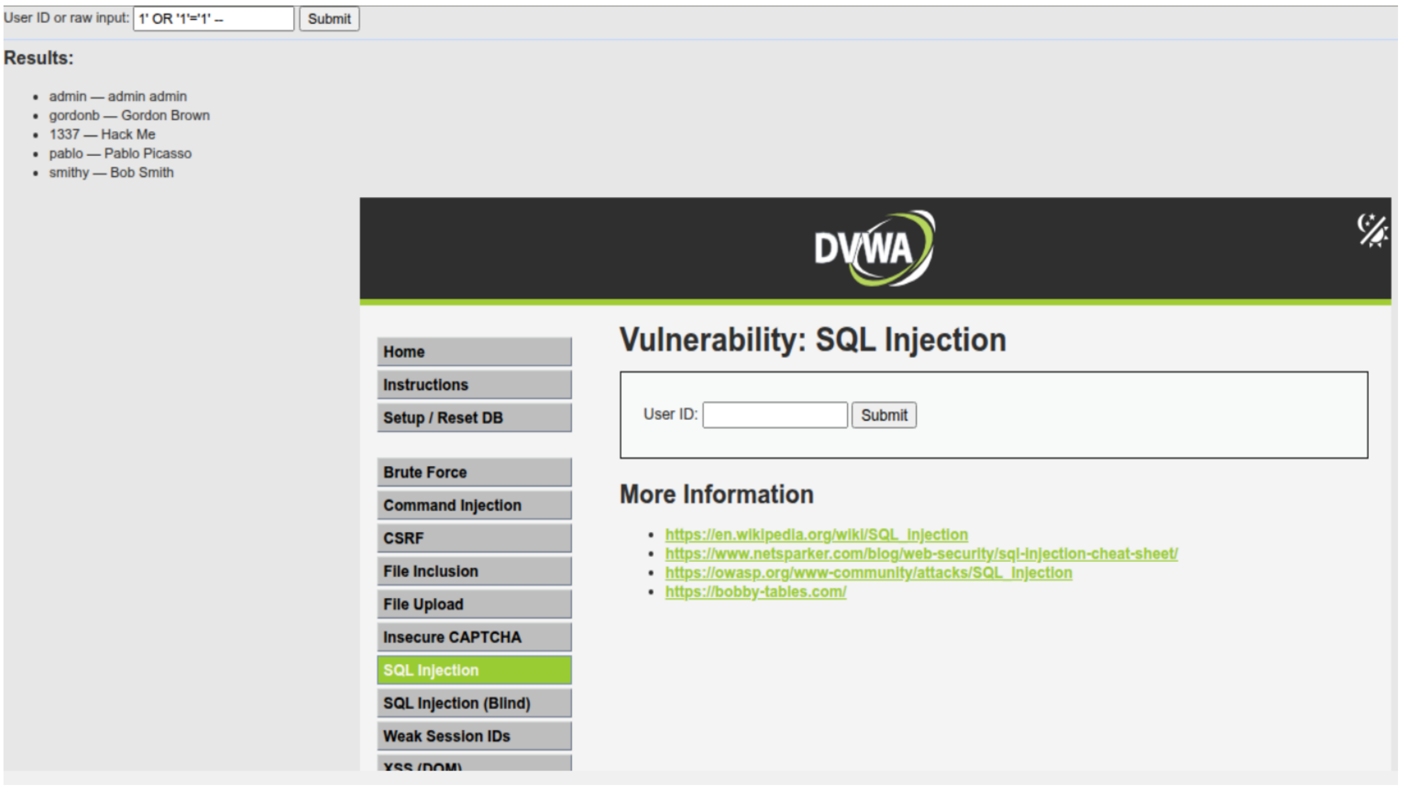
Security Testing Report

OWASP stands for Open Web Application Security Project.
One of its most well-known contributions is the ‘OWASP Top 10,’ which is a list of the 10 most critical web application security risks
Understanding OWASP Top 10: DVWA has examples of multiple vulnerabilities like:

- SQL Injection (SQLi)
- Cross-Site Scripting (XSS)
- CSRF (Cross-Site Request Forgery) SQL Injection

1. SQL Injection is a vulnerability where an attacker can manipulate a web application’s SQL queries by injecting malicious input.

- When user input is directly inserted into SQL queries without validation or escaping.
- The database executes whatever the input modifies the query to do.
- STEPS:
- Go to the SQL Injection page:
- Input box ID takes user input. Example: 1 OR 1=1
- Input manipulates the query to: User_id = '1' OR 1=1; It always evaluates the query to true → all users are returned
- Output: You see multiple usernames and passwords on the page.
- Due to this Attackers can read sensitive data, modify/delete records, or even compromise the system.
- Fix- Demostrate Prepared Statement so that only correct user_id rows are returned.



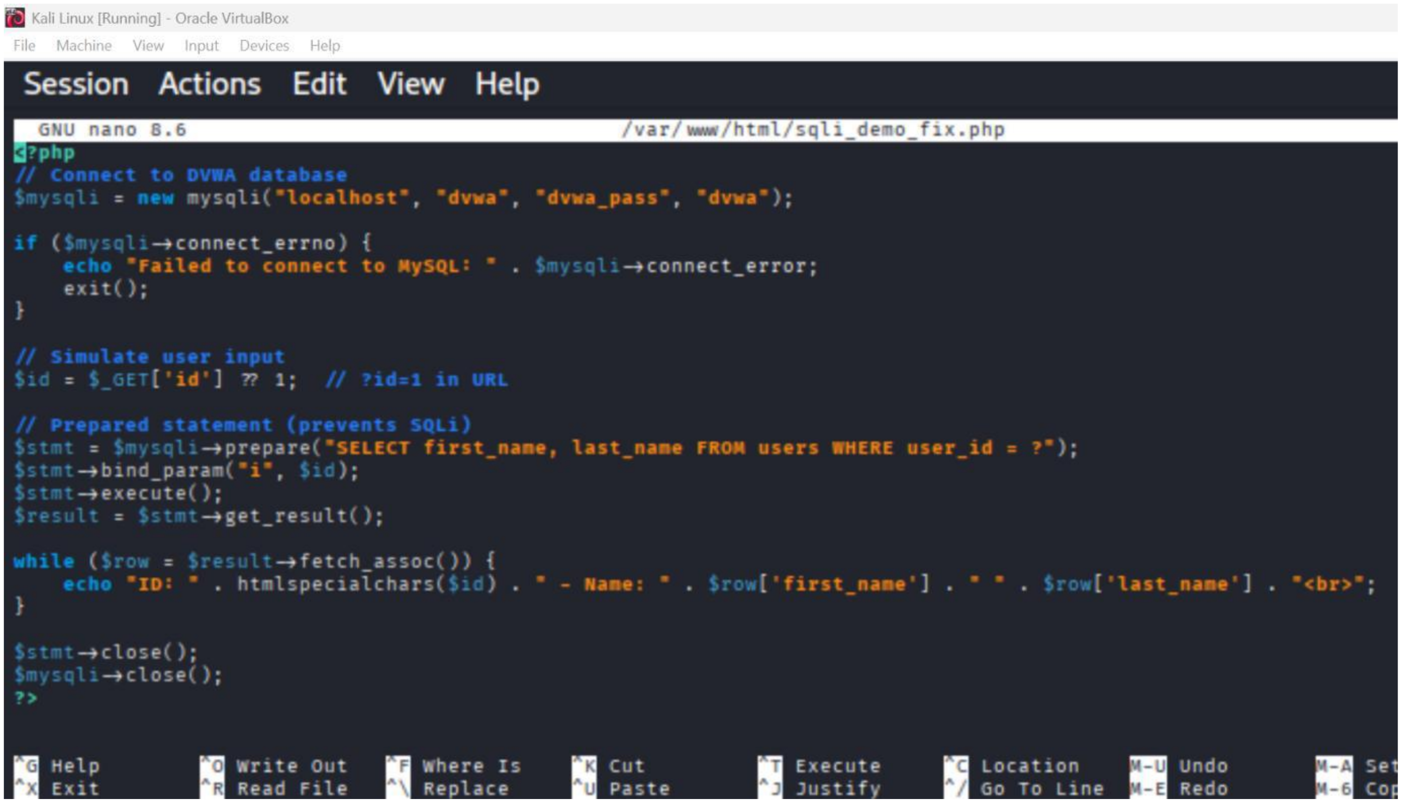
User ID or raw input: 1' OR '1'='1' --

Results:

- admin — admin admin
- gordonb — Gordon Brown
- 1337 — Hack Me
- pablo — Pablo Picasso
- smithy — Bob Smith

DVWA
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)
Vulnerability: SQL Injection
User ID:
Submit
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/

Demonstrating the Fix (Prepared Statements)

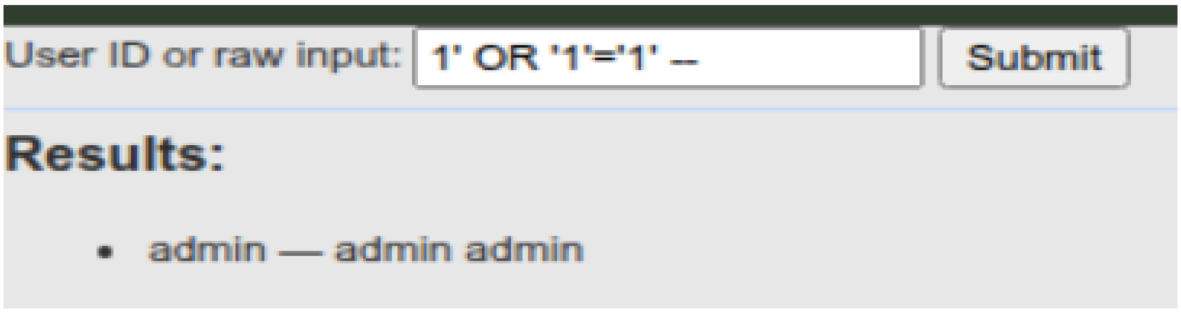

Kali Linux [Running] - Oracle VirtualBox
File Machine View Input Devices Help
Session Actions Edit View Help

```
GNU nano 8.6                                        /var/www/html/sqli_demo_fix.php
<?php
// Connect to DVWA database
$mysqli = new mysqli("localhost", "dvwa", "dvwa_pass", "dvwa");

if ($mysqli→connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli→connect_error;
    exit();
}

// Simulate user input
$id = $_GET['id'] ?? 1;  // ?id=1 in URL

// Prepared statement (prevents SQLi)
$stmt = $mysqli→prepare("SELECT first_name, last_name FROM users WHERE user_id = ?");
$stmt→bind_param("i", $id);
$stmt→execute();
$result = $stmt→get_result();

while ($row = $result→fetch_assoc()) {
    echo "ID: " . htmlspecialchars($id) . " - Name: " . $row['first_name'] . " " . $row['last_name'] . "<br>";
}

$stmt→close();
$mysqli→close();
?>
```

^G Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Cop

Prepared statements separate query structure from user input:
$stmt = $db->prepare("SELECT * FROM users WHERE user_id = ?");
$stmt->bind_param("i", $id);
? is a placeholder; $id is bound safely.
Even if the user enters '1 OR 1=1', the database treats it as a literal value, not part of SQL.
It only provides the first id as output.

Prepared Statement Output:



Results:

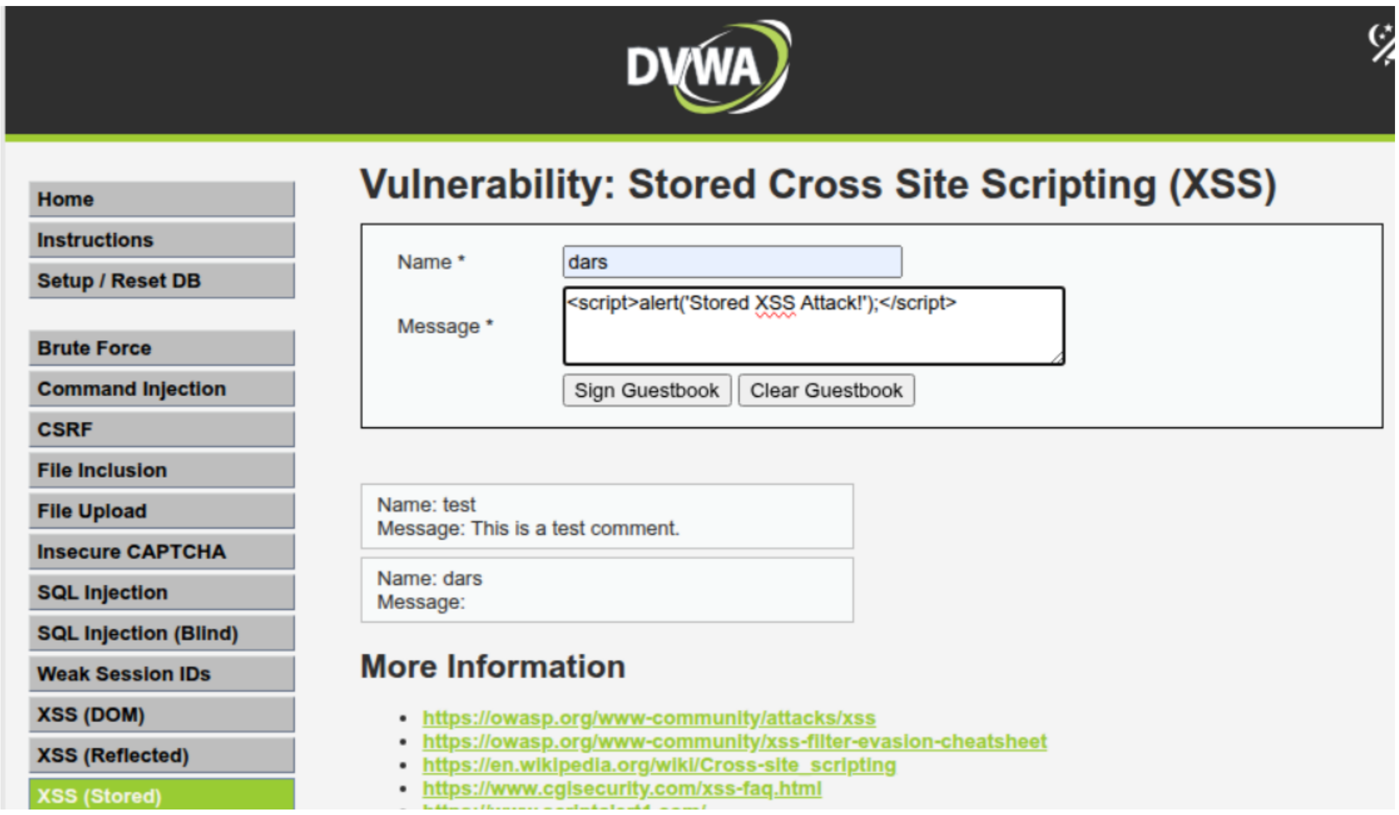
- admin — admin admin

Result: Injection fails → only correct user_id rows are returned.

2. **Cross-Site Scripting (XSS)** is a class of web vulnerability where an attacker is able to inject client-side scripts (usually JavaScript) into pages viewed by other users. When the victim's browser executes the injected script, the attacker can do things like steal session cookies, manipulate the DOM, perform actions as the user, show fake UI, etc.
Two common types:

- **Stored (Persistent) XSS**

How it works: Attacker submits malicious script into a site input (comment, profile, message) that the server stores in the database. When any user later views that stored data, the script runs in their browser.
- Impact: Broad — any user who views the page can be affected; can be used to steal cookies, perform actions, pivot.
- Consequences of Stored XSS include:
- Redirecting users to malicious sites
- Stealing cookies or session tokens
- Defacing the website
- Performing actions on behalf of other users
Commands
- Open DVWA and navigate to Stored XSS page:
- Enter malicious payload in the Guestbook form:
- Name: Dars
- Message: <script>alert('Stored XSS Attack!');</script>
- Click Sign Guestbook



Vulnerability: Stored Cross Site Scripting (XSS)

Name * dars
Message * <script>alert('Stored XSS Attack!');</script>

Name: test
Message: This is a test comment.

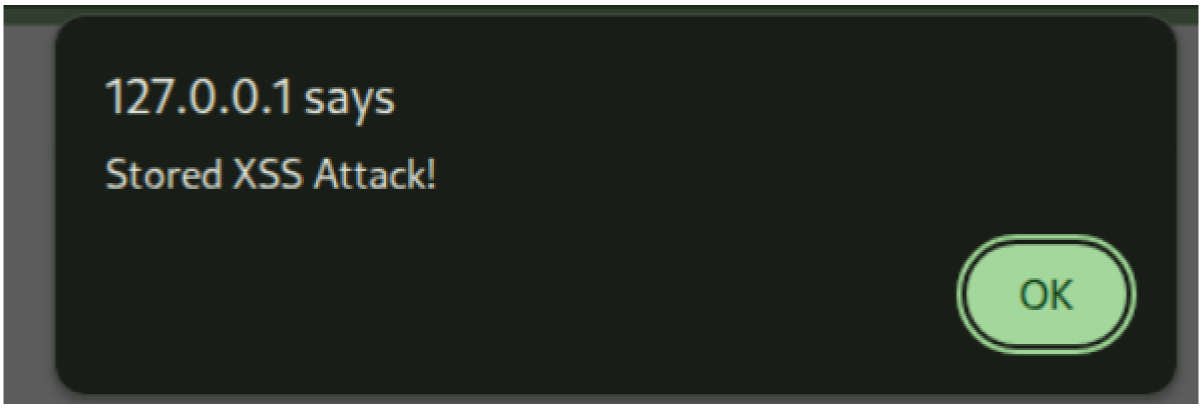
Name: dars
Message:

More Information

- 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://www.cgisecurity.com/xss-faq.html

Output
- The page reloads and shows the guestbook entries.
- A popup with Stored XSS Attack appears — this confirms that the browser executed the injected script.
- When any user loads that page, the malicious code executes in their browser.

- In our lab, the popup Stored XSS Attack is just a safe demonstration — it proves that the input was executed as code instead of being treated as plain text.
- In real life, a hacker could replace Stored XSS Attack with something harmful: stealing cookies, logging keystrokes, redirecting the user, etc.



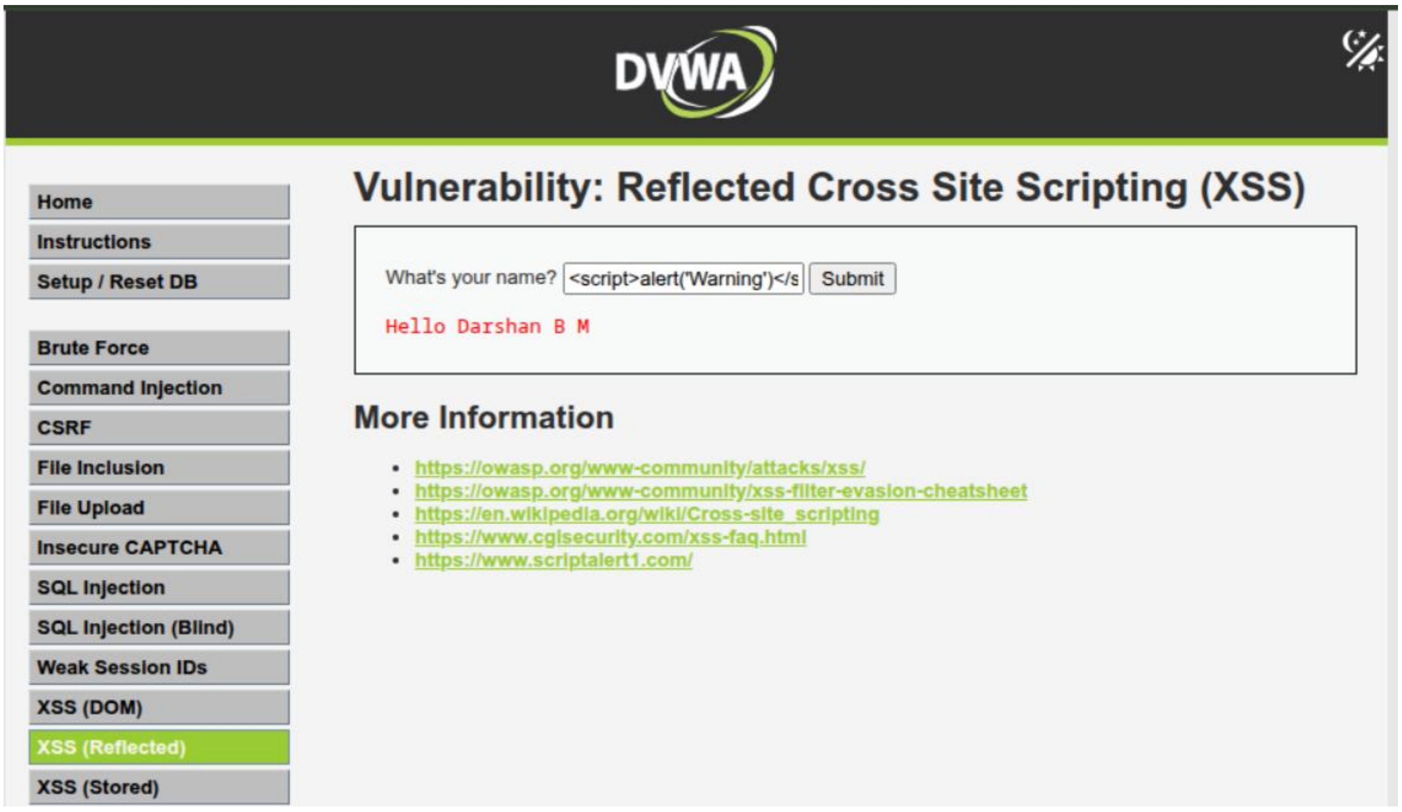
127.0.0.1 says

Stored XSS Attack!

- **Reflected XSS**

How it works: The server reflects attacker-controlled input immediately in a response (e.g., search term shown in results or a name parameter echoed back), without storing it. The attacker crafts a URL containing the payload and convinces a victim to click it.
Impact: Targeted — requires tricking a victim (phishing link), but still very dangerous.
- Consequences include:
- Stealing cookies/session tokens
- Phishing attacks
- Redirection to malicious websites

Commands:
- Open DVWA and navigate to Reflected XSS page
- Your name: <script>alert('Warning')</script>



Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? <script>alert('Warning')</s

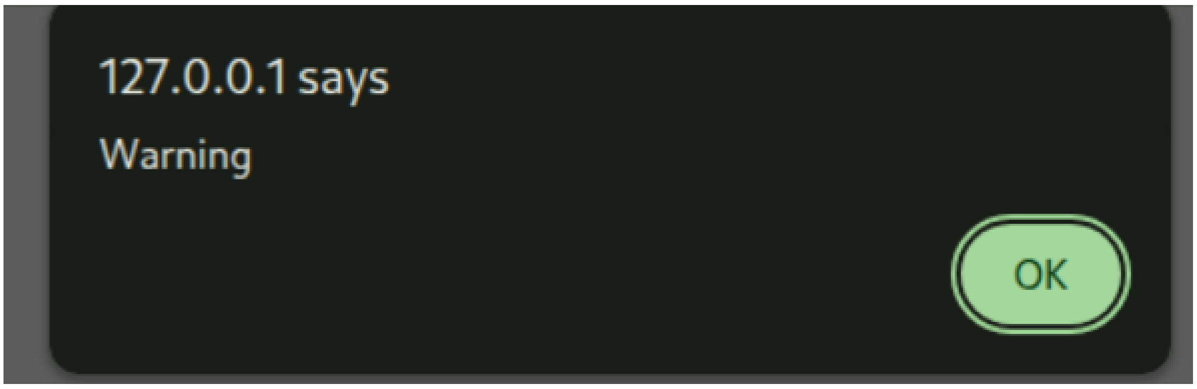
Hello Darshan B M

More Information

- 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://www.cgisecurity.com/xss-faq.html
- https://www.scriptalert1.com/

Output
- The input was reflected directly in the page response.
- The browser executed it as JavaScript, producing the alert popup.

- This demonstrates a Reflected XSS vulnerability, showing that any user who clicks the link could trigger the script.
- Attackers exploit it via malicious links in emails, social media, or phishing pages.



127.0.0.1 says

Warning

How to prevent XSS:

A. Input Sanitization / input validation (server-side)/ WhiteListing:

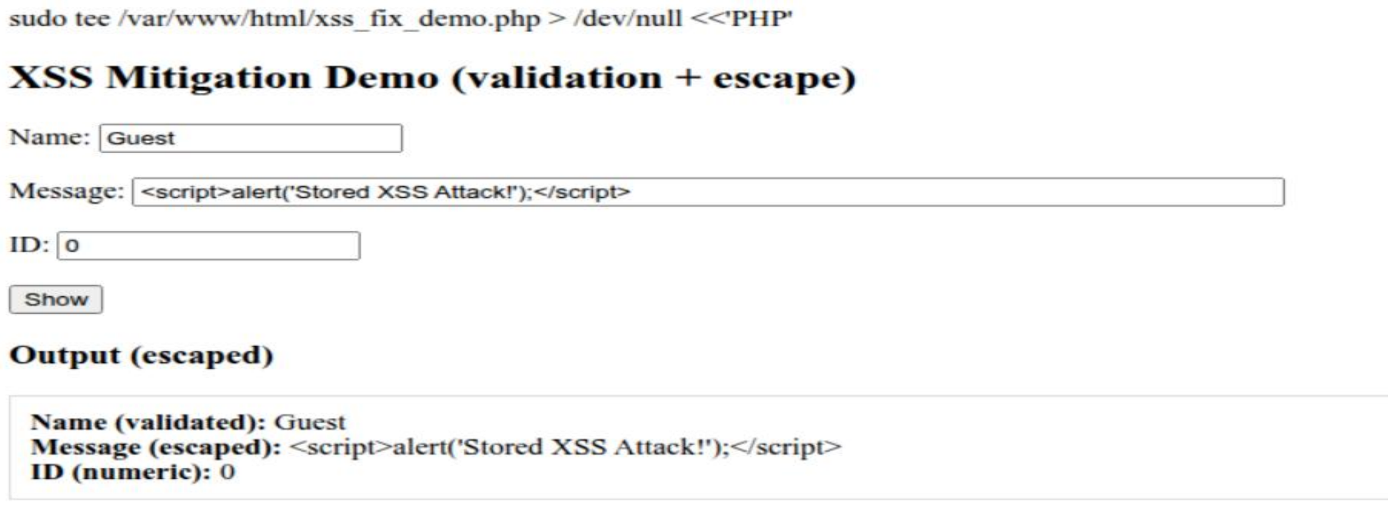
Checking and restricting what users can input before it goes to the database or page.Restrict what users can input. Example: allow only letters/numbers for a name field. Remove or encode HTML special characters (<, >, &, ", '). This ensures scripts are displayed as text, not executed.
Malicious users often inject scripts via form fields. If we restrict inputs to safe characters, scripts cannot execute. Displayed as text, not executed.
- Demo: Simple whitelist for name and numeric casting for id
- Creating demo file or a small processing script.
- Eg. Demo file xss_fix_demo.php showing how to clean name and how to cast an id:
Explanation of the validation lines in the below picture:

- preg_replace('/[^a-zA-Z0-9 _-]/', '', $name_raw);
- This removes any character that is not A-Z, a-z, 0-9, space, underscore, or hyphen. It's a whitelist approach: only allowed chars remain.
- is_numeric($id_raw) ? (int)$id_raw : 0;
- If id_raw is numeric, cast to integer; otherwise set to 0. This prevents injecting SQL-injection-like values into ID fields.

Why validate on input?

- Reduces attack surface (if names are only letters, a <script> tag is removed).
- Useful for fields that should be constrained (IDs, dates, phone numbers).

Output:
1. Name will be stripped of <script> and likely become alert1 or empty (depending on characters).



```
sudo tee /var/www/html/xss_fix_demo.php > /dev/null <<'PHP'
```

XSS Mitigation Demo (validation + escape)

Name: Guest

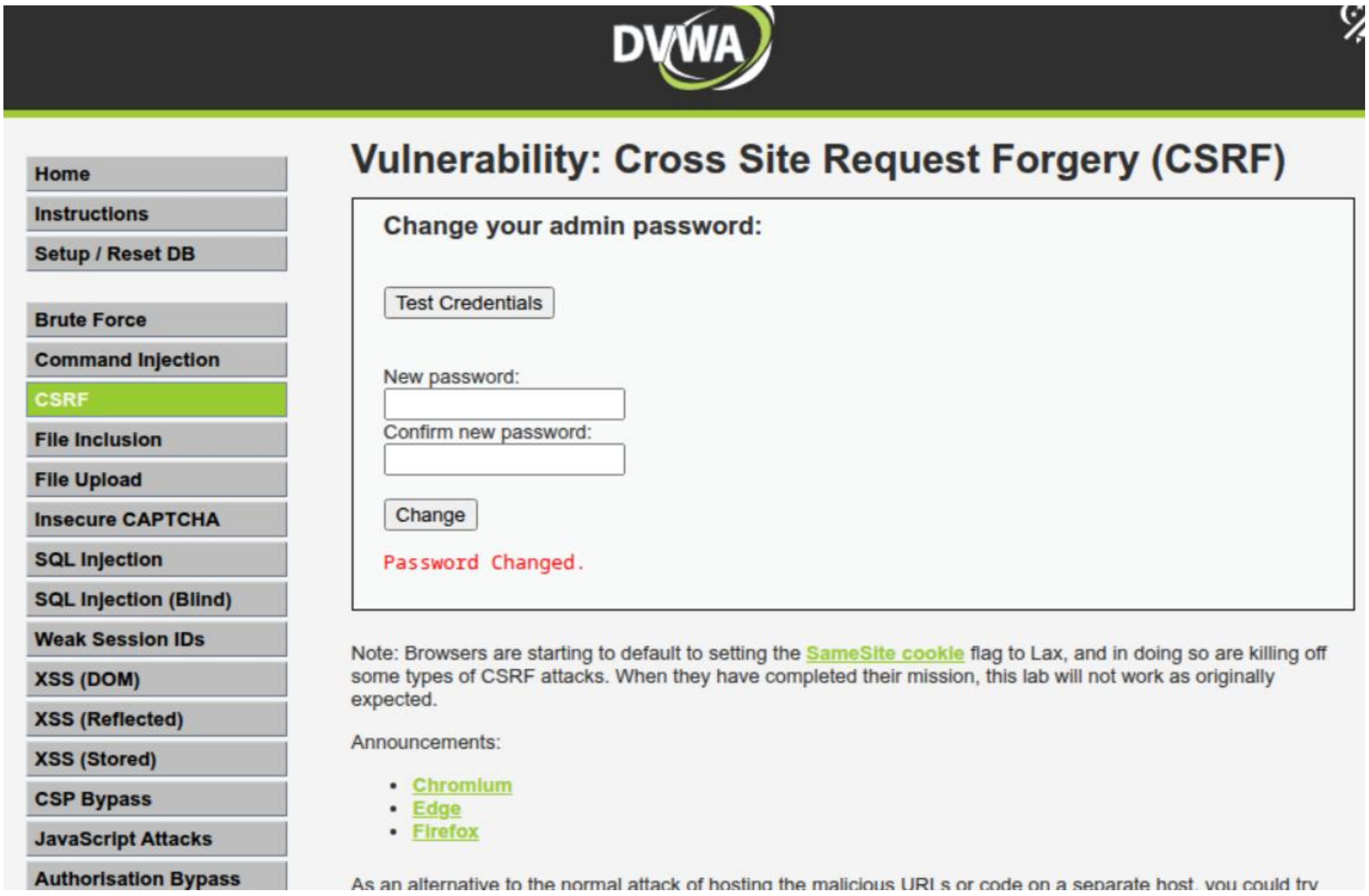
Message: <script>alert('Stored XSS Attack!');</script>

ID: 0

Output (escaped)

Name (validated): Guest
Message (escaped): <script>alert('Stored XSS Attack!');</script>
ID (numeric): 0

3. CSRF (Cross-Site Request Forgery) is a type of web security vulnerability.
It tricks a logged-in user into unknowingly performing actions on a website (like changing their password, transferring money, deleting an account) without their consent.How it Works
- You are logged in to DVWA (or any site) in one tab.
- The site uses your session cookie to know you are authenticated.
- Attacker sends you a malicious link or form.
- It contains this kind of exploit code.



Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

New password:
Confirm new password:

Password Changed.

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try

Why is it Dangerous? Why is it Dangerous?
- No user interaction needed except clicking/opening attacker’s page.
- Works silently because cookies are sent automatically by the browser.
- Can lead to account takeover.

How to Prevent (Mitigation)
- The most common protection is CSRF Token (anti-CSRF token):
- Website generates a random unique token for each form request.
- The token is stored in the user’s session.
- When form is submitted, the token is validated.
- If token is missing/wrong → request is blocked.
- Switched DVWA Security From Low to High, which added a per-session hidden token to the CSRF form (e.g. <input type="hidden" name="user_token" value="...">).