Reporting Practice

Executive Summary

A security assessment was conducted on the web application hosted at **http://192.168.0.12**. Multiple high-risk vulnerabilities were identified, including SQL Injection, Command Injection, weak authentication controls, insecure file upload handling, and missing security headers. These weaknesses allow attackers to bypass login, execute system commands, upload malicious files, and potentially take full control of the server. Automated scans using OWASP ZAP validated these findings. Immediate remediation is required to prevent unauthorized access, data exposure, and system compromise. Strengthening authentication, input validation, file upload restrictions, and server configuration will greatly improve overall security.

2. TECHNICAL FINDINGS

This section documents each vulnerability with technical details, exploitation method, risk rating, evidence, and recommended fixes.

Finding F001 – SQL Injection (Login – Manual)

Description:

The login form does not properly sanitize user-supplied input, allowing direct manipulation of backend SQL queries.

Impact:

Unauthorized login, full database exposure, privilege escalation, admin account takeover.

CVSS Score: 9.1 (Critical)

Evidence:

Payload used:

admin' OR 1=1-- -

Result: Successful authentication without valid credentials.

Likelihood: High

Finding F002 – SQL Injection (Login – sqlmap)

Description:

Automated testing confirms SQLi vulnerability in the login POST request. sqlmap was able to enumerate databases, tables, and user hashes.

Impact:

Database dump, account compromise, modification or deletion of data.

CVSS Score: 9.1 (Critical)

Evidence:

sqlmap output:

Databases: dvwa, information_schema

Tables: users

Dumped: Usernames + password hashes

Likelihood: High

Finding F003 – SQL Injection (GET Parameter)

Description:
The page /vulnerabilities/sqli/ is vulnerable to SQL injection through GET parameters without any input filtering.
Impact:
Database access, extraction of sensitive information, authentication bypass.
CVSS Score: 9.0 (Critical)
Evidence:
Payload:
id=1' OR '1'='1
Returned valid user data and no errors.
Likelihood: High

Finding F004 – Command Injection (exec module)

Description:
The command execution module accepts unsanitized input, allowing execution of system-level commands on the server.
Impact:
Remote Code Execution (RCE), privilege escalation, full server compromise.
CVSS Score: 9.5 (Critical)
Evidence:
Payloads:
; whoami
&& ls
Returned valid OS command output.
Likelihood: Very High

Finding F005 – File Upload Bypass → RCE

Description:
The upload module allows uploading of PHP scripts disguised as images, enabling remote code execution.
Impact:
Webshell upload, full server takeover, persistence, internal network access.
CVSS Score: 9.8 (Critical)
Evidence:
Uploaded explot.php.jpg → change to exploit.php → Executed via web browser → Returned system commands.
Likelihood: Very High

Finding F006 – Weak Authentication (Password Guessing)

Description:
Login mechanism lacks brute-force protection and uses default/weak credentials.
Impact:
Account takeover, unauthorized access.
CVSS Score: 7.5 (High)
Evidence:
Successful login using default passwords:
admin:admin
Likelihood: Medium-High

Finding F007 – Insecure Cookie Flags

Description:
PHPSESSID cookie lacks Secure, HttpOnly, and SameSite attributes.
Impact:
Risk of session hijacking, XSS-based cookie theft, MITM interception.
CVSS Score: 7.0 (High)
Evidence:
Header:
Set-Cookie: PHPSESSID=" "; path=/
Likelihood: Medium

Finding F008 – OWASP ZAP Active Scan Findings

Description:
Automated scanning identified multiple weaknesses including missing headers, outdated components, and input fields lacking validation.
Impact:
Increased exposure to attacks such as XSS, clickjacking, MITM.
CVSS Score: 7.0 (High)
Evidence:
ZAP alerts:
Missing Security Headers
Weak CSP
Potential XSS points
Likelihood: Medium

Finding F009 – Reflected XSS

Description:

Input is reflected back to the browser without proper encoding or filtering.

Impact:

Session hijacking, credential theft, defacement, phishing.

CVSS Score: 6.1 (Medium)

Evidence:

Payload displayed unescaped:

<script>alert('XSS')</script>

Likelihood: Medium

Finding F010 – Missing Security Headers

Description:

Server responses lack industry-standard security headers.

Impact:

Susceptible to clickjacking, MIME attacks, MITM, XSS.

CVSS Score: 6.5 (Medium)

Evidence:

Missing:

- Content-Security-Policy
- Strict-Transport-Security
- X-Frame-Options
- X-Content-Type-Options

Likelihood: Medium

3. FINDINGS TABLE

Findings:

Finding ID	Vulnerability	CVSS Score	Severity	Target URL
F001	SQL Injection – Login (Manual)	9.1	Critical	/login.php
F002	SQL Injection – Login (sqlmap)	9.1	Critical	/login.php
F003	SQL Injection – GET Parameter	9.0	Critical	/vulnerabilities/sqli/
F004	Command Injection (exec module)	9.5	Critical	/vulnerabilities/exec/
F005	File Upload Bypass → RCE	9.8	Critical	/vulnerabilities/upload/
F006	Weak Authentication (Password Guessing)	7.5	High	/login.php
F007	Insecure Cookie Flags	7.0	High	/http://192.168.0.12
F008	OWASP ZAP Active Scan Findings	7.0	High	/http://192.168.0.12
F009	XSS – Reflected	6.1	Medium	/vulnerabilities/xss_r/
F010	Missing Security Headers	6.5	Medium	All responses

4. REMEDIATION PLAN

Priority: Critical

1. Implement Parameterized Queries

All SQL database interactions must use parameterized queries or prepared statements to completely eliminate SQL injection vulnerabilities. Replace all dynamic string concatenation with safe query execution frameworks.

2. Deploy a Web Application Firewall (WAF)

Implement a WAF (e.g., ModSecurity, AWS WAF, Cloudflare WAF) to detect and block SQLi, XSS, RCE, and command injection patterns before they reach the application.

3. Validate and Restrict File Uploads

Enforce strict file upload controls:

- Validate MIME types and extensions
- Allow only safe formats (jpg, png, pdf)
- Block all executable content (php, jsp, exe)
- Perform server-side scanning
- Store uploads outside the webroot

This prevents malicious files from being executed on the server.

4. Sanitize Shell Inputs and Disable Dangerous Functions

Completely remove system command execution through user input.

Disable high-risk PHP functions such as:

exec(), system(), passthru(), shell_exec(), eval()

Priority: High

5. Strengthen Authentication

Implement secure authentication controls:

- Minimum 12-character passwords
- Require uppercase, lowercase, numbers, and symbols
- Enforce account lockout after repeated failed attempts
- Disable default credentials

6. Enable Multi-Factor Authentication (MFA)

Add an additional authentication layer (TOTP, SMS, or app-based) to prevent unauthorized access even if credentials are compromised.

7. Secure Cookie and Session Management

Apply secure cookie attributes:

- HttpOnly
- Secure
- SameSite=Strict

Regenerate sessions after login and enforce session timeout policies.

8. Restrict Directory Access

Prevent execution in upload directories:

- Set upload folder permissions to **read/write only**
- Disable script execution using .htaccess or server config

This blocks webshell execution.

Priority: Medium

9. Remove Verbose Error Messages

Disable detailed server errors and warnings that expose:

- Database structure
- File paths
- Server information

Show only generic error messages to end users.

10. Apply Security Headers

Add the following headers to reduce XSS, clickjacking, and MITM risks:

- Content-Security-Policy
- Strict-Transport-Security
- X-Frame-Options
- X-Content-Type-Options
- Referrer-Policy

11. Regular Security Patching

Ensure the operating system, web server, PHP modules, and database engine are updated regularly to close known vulnerabilities.

12. Implement Least-Privilege Access

Ensure service accounts and users operate with minimal required privileges:

- Remove unnecessary admin rights
- Isolate application processes
- Segregate database permissions

This reduces potential damage from exploitation.

5. 3×3 RISK ASSESSMENT MATRIX

Legend:

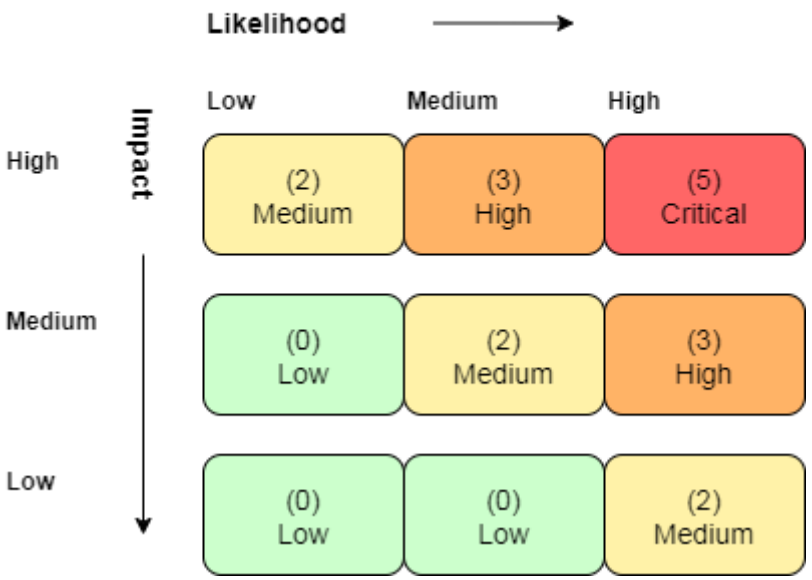
- **H = High**
- **M = Medium**
- **L = Low**
- **Risk Levels:** Low, Medium, High, Critical

Impact ↓ / Likelihood →	**Low (L)**	**Medium (M)**	**High (H)**
High Impact (H)	Medium Risk	High Risk	Critical Risk
Medium Impact (M)	Low Risk	Medium Risk	High Risk
Low Impact (L)	Low Risk	Low Risk	Medium Risk

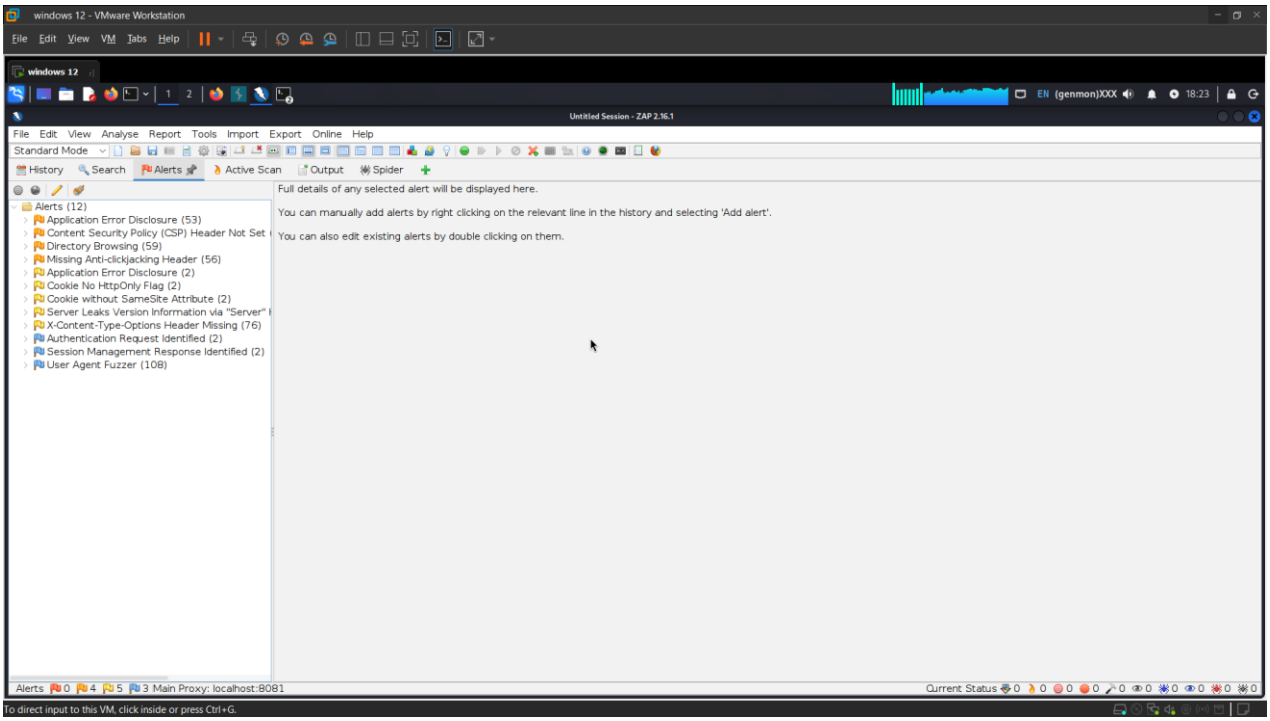
Mapping Findings to Risk Levels

Finding ID	**Vulnerability**	**Impact**	**Likelihood**	**Risk Level**
F001	SQL Injection – Login (Manual)	High	High	Critical
F002	SQL Injection – Login (sqlmap)	High	High	Critical
F003	SQL Injection – GET Parameter	High	High	Critical
F004	Command Injection (exec)	High	High	Critical
F005	File Upload Bypass → RCE	High	High	Critical
F006	Weak Authentication (Password Guessing)	Medium	Medium	Medium
F007	Insecure Cookie Flags	Medium	Medium	Medium
F008	OWASP ZAP Active Scan Findings	Medium	Medium	Medium
F009	XSS – Reflected	Medium	Medium	Medium
F010	Missing Security Headers	Medium	Medium	Medium

3×3 Risk Assessment Matrix



OWASP ZAP Alert



Screenshot:OWSAP ZAP Alerts

OWASP ZAP Alert Summary
High Alerts (0)
(No high-severity alerts detected)

🔶 **Medium Alerts (4)**

1. **Application Error Disclosure (53)**
2. **Content Security Policy (CSP) Header Not Set (57)**
3. **Directory Browsing (59)**
4. **Missing Anti-Clickjacking Header (56)**

🔵 **Low Alerts (5)**

5. **Application Error Disclosure (2)**
6. **Cookie No HttpOnly Flag (2)**
7. **Cookie without SameSite Attribute (2)**
8. **Server Leaks Version Information via "Server" Header (1)**
9. **X-Content-Type-Options Header Missing (76)**

🟢 **Informational Alerts (3)**

10. **Authentication Request Identified (2)**
11. **Session Management Response Identified (2)**
12. **User Agent Fuzzer (108)**

Your screenshot = Total: 12 alerts

- **High:** 0
- **Medium:** 4
- **Low:** 5
- **Info:** 3

6. NETWORK ATTACK PATH DIAGRAM

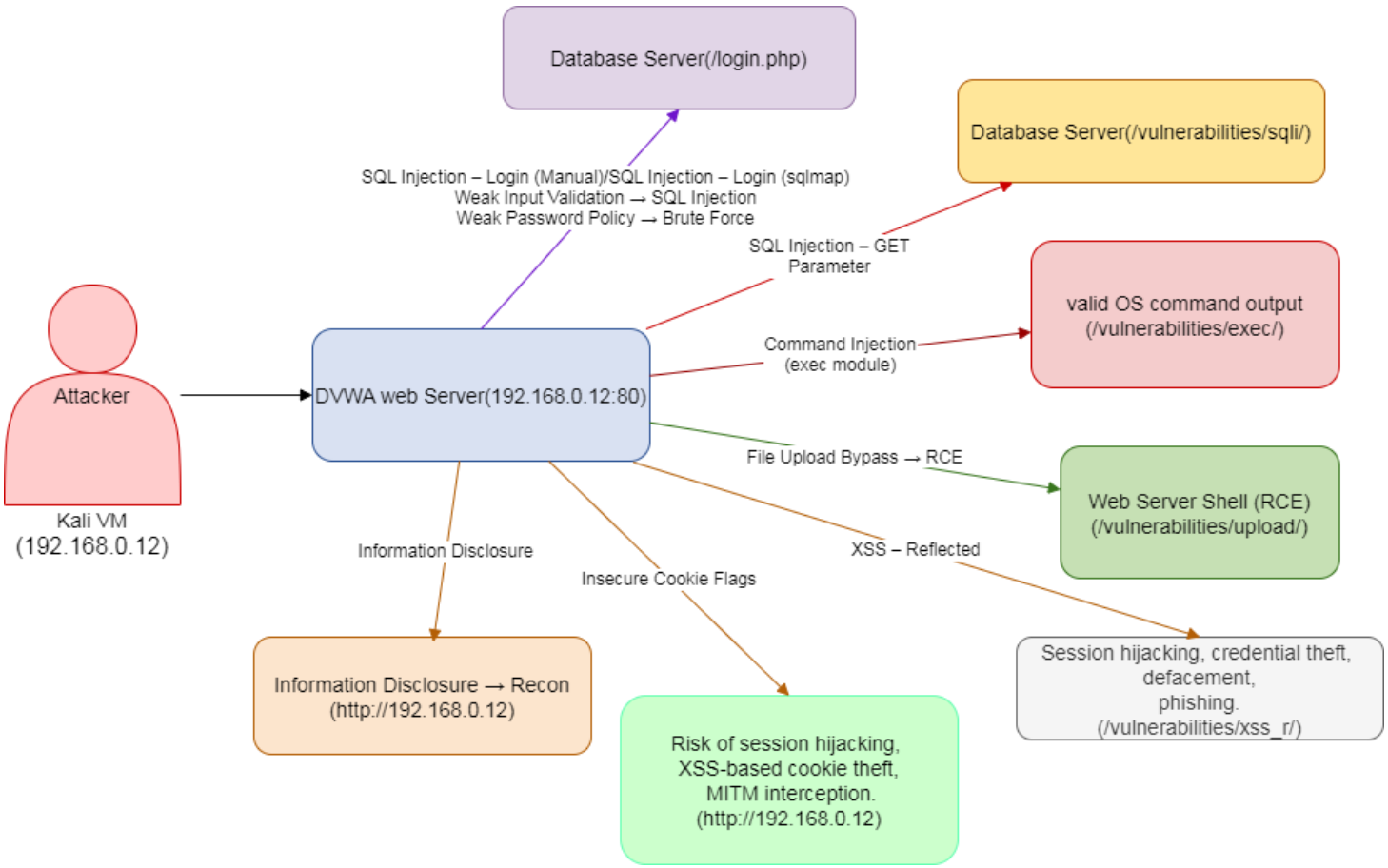


Fig: Network Diagram

7. SUMMARY

The security test revealed several high-risk weaknesses in the web application that could allow attackers to gain unauthorized access, steal sensitive information, or take control of the system. Critical issues such as unsafe login mechanisms, weak file upload controls, and poor input validation create opportunities for complete system compromise. High-severity weaknesses such as insecure cookies and missing authentication controls further add to the risk. Medium-level issues like missing security headers also reduce overall protection. Immediate action is recommended to secure the application and prevent potential misuse or data breaches.