1. OWASP Top 10 Summary with Examples

S.No	Vulnerability	Explanation	Example
1	Broken Access Control	Users access data or functions they shouldn't	/admin accessible by regular user
2	Cryptographic Failures	Weak or no encryption on sensitive data	Passwords sent over HTTP
3	Injection (SQLi)	Malicious queries via unsanitized input	' OR '1'='1 in login field
4	Insecure Design	Poorly designed security in system architecture	No brute-force protection
5	Security Misconfiguration	Default settings or verbose errors	Displaying stack trace in production
6	Vulnerable & Outdated Components	Using unpatched libraries or plugins	jQuery 1.x with known XSS bugs
7	ID & Authentication Failures	Flawed login/session mechanisms	No 2FA, weak password policies
8	Software/Data Integrity Failures	Unverified updates, insecure CI/CD	Auto-updating plugin from unknown source
9	Logging & Monitoring Failures	No attack detection or alert system	No logs for failed logins
10	Server-Side Request Forgery (SSRF)	Server requests internal resources	http://localhost:8000/admin accessed via app

2. XSS and SQLi Testing on DVWA

Tools Used: DVWA, Burp Suite

Security Level: Low

XSS Payload:

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

SQL Injection Payload:

```
' OR '1'='1' --
```

Result: Login bypassed. Burp Suite used for interception and injection.

3. Input Validation & Code Review

Sample 1: Vulnerable PHP Script

```
$username = $_POST['user'];
$password = $_POST['pass'];
$sql = "SELECT * FROM users WHERE user='$username' AND pass='$password'";
```

Fixed Version:

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE user = ? AND pass = ?");
$stmt->execute([$username, $password]);
```

Sample 2: JavaScript Injection Risk

```
<input type="text" name="email" id="email">
<script>
let email = document.getElementById("email").value;
document.write("Welcome " + email);
</script>
```

Fixed Version:

```
let email = document.getElementById("email").value;
document.write("Welcome " + escape(email));
```

4. Security Headers & HTTPS

Analyzed Site: example.com (via securityheaders.com)

Missing Headers:

- Content-Security-Policy

- Strict-Transport-Security

- X-Frame-Options

- X-Content-Type-Options

Apache Fix:

```
Header always set Content-Security-Policy "default-src 'self'; script-src 'self';"
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
Header set X-Frame-Options "DENY"
Header set X-Content-Type-Options "nosniff"
```

5. Bonus: Simple WAF Rule Setup (ModSecurity)

WAF Rule to Block SQL Injection Patterns:

```
SecRule              ARGS              "@rx              (\bUNION\b|\bSELECT\b|\bDROP\b)"
"id:1234,phase:2,deny,status:403,msg:'SQLi Attempt Detected'"
```

Testing: Injection blocked with 403 Forbidden.