

# Module 5 — Ethical Hacking (Viva Questions & Answers)

## Q1: What is Cross-Site Scripting (XSS)?

A1: XSS is a vulnerability where an application includes untrusted user input in web pages without proper sanitization, allowing attackers to execute JavaScript in victims' browsers. Types: Stored, Reflected, DOM-based.

## Q2: How to prevent XSS?

A2: Encode output contextually (HTML, JS, attribute), use Content Security Policy (CSP), sanitize input where appropriate, and use frameworks that auto-escape templates.

## Q3: What is Cross-Site Request Forgery (CSRF)?

A3: CSRF tricks authenticated users into performing unwanted actions on a web app by exploiting their active session (e.g., submitting forms). It leverages victims' credentials without their intent.

## Q4: Defenses against CSRF?

A4: Use anti-CSRF tokens (per-session/request), require same-site cookies (SameSite=strict/lax), and verify Origin/Referer headers for sensitive requests.

## Q5: What is SQL Injection in web apps and a simple prevention method?

A5: SQLi allows attackers to inject SQL code via unsanitized inputs. Prevent with parameterized queries (prepared statements), ORM safe APIs, and input validation.

## Q6: Explain Remote File Upload vulnerability and mitigation.

A6: Unsafe file upload can let attackers upload web shells. Mitigate by validating file types server-side, store uploads outside webroot, rename files, set strict permissions, and scan for malware.

## Q7: What is Remote Code Execution (RCE) and common causes?

A7: RCE allows attackers to execute system commands or code on the server. Causes: unsanitized eval, unsafe deserialization, shelling user inputs, vulnerable native libraries.

## Q8: What is insecure deserialization?

A8: Insecure deserialization occurs when untrusted serialized data is deserialized without validation, allowing attackers to instantiate objects, run code, or escalate privileges.

## Q9: How to secure APIs (REST/GraphQL)?

A9: Use strong authentication/authorization (OAuth2, JWT with care), rate limiting, input validation, strict CORS policies, pagination, and logging. Validate JSON schema and avoid leaking error details.

## Q10: What are JWT risks and best practices?

A10: Risks: weak signing keys, none algorithm misuse, token theft. Best practices: use strong keys, validate signatures and expiry, rotate keys, store tokens securely (httpOnly cookies), and check revocation where needed.

## Q11: What is Server-Side Request Forgery (SSRF)?

A11: SSRF tricks a server into making requests to internal or external resources (e.g., cloud metadata endpoints), potentially exposing sensitive data or enabling pivoting.

## Q12: How to mitigate SSRF?

A12: Implement allowlists for outbound URLs/IPs, validate and canonicalize URLs, block private IP ranges, use network egress controls, and avoid fetching arbitrary user-supplied URLs.

## Q13: Explain XML External Entity (XXE) attacks.

A13: XXE exploits poorly configured XML parsers to process external entities, allowing file disclosure, SSRF, or remote interaction. Disable external entities and use secure parsers.

**Q14: What is Clickjacking and prevention techniques?**

A14: Clickjacking overlays pages to trick users into clicking hidden controls. Prevent with X-Frame-Options (DENY/SAMEORIGIN) or CSP frame-ancestors directives.

**Q15: Describe Directory Traversal and protection.**

A15: Directory traversal lets attackers read files outside intended directories via ../ input. Prevent by canonicalizing paths, enforcing allowed directories, and validating filenames.

**Q16: Why is input validation important and what is a whitelist approach?**

A16: Input validation prevents many injection flaws. Whitelist (positive validation) accepts only expected values/patterns, e.g., allowed file extensions or numeric ranges, instead of blacklisting.

**Q17: What security headers should a web app set?**

A17: Content-Security-Policy, X-Frame-Options, X-Content-Type-Options: nosniff, Strict-Transport-Security (HSTS), Referrer-Policy, and Feature-Policy (Permissions-Policy).

**Q18: How does CORS work and what misconfigurations to avoid?**

A18: CORS controls cross-origin browser requests. Avoid wildcard origins with credentials, and explicitly allow only trusted origins and methods. Validate preflight handling.

**Q19: Tools for web application testing?**

A19: Burp Suite, OWASP ZAP, Nikto, sqlmap, wfuzz, DirBuster/dirsearch, and browser developer tools. Use automated scanners plus manual testing for logic flaws.

**Q20: What is a Web Application Firewall (WAF) and its limits?**

A20: WAF inspects HTTP(S) traffic to block common attacks. It's useful for protection and virtual patching, but can't replace secure coding and can be bypassed by sophisticated or novel attacks.

**Q21: How to secure file permissions and uploads on servers?**

A21: Run services with least privilege accounts, set uploads to non-executable directories, use proper umask, and isolate via containers or chroot. Scan/uploads and enforce size limits.

**Q22: What is privilege separation and process isolation?**

A22: Separating privileges limits damage by running components with minimal rights; isolation (containers, VMs) reduces blast radius and prevents cross-service compromise.

**Q23: What is Threat Modeling in web app security?**

A23: Threat modeling identifies assets, threat actors, attack surfaces, and countermeasures—using methods like STRIDE or DREAD—to prioritize security work.

**Q24: Explain SSR (Server Side Rendering) vs CSR (Client Side Rendering) security considerations.**

A24: SSR can leak sensitive server data in templates; ensure proper escaping. CSR shifts logic to client—be cautious with exposing secrets and rely on secure APIs and token handling.

**Q25: What should a vulnerability report for web app include?**

A25: Summary, affected endpoints, vulnerability type, severity, proof of concept (request/response, screenshots), remediation steps, risk impact, and remediation verification guidance.