SECURE APPLICATION DEVELOPMENT LAB VIVA

Q1. What is OWASP?

OWASP stands for *Open Web Application Security Project*. It is a non-profit community that provides resources and guidelines for developing secure web applications.

Q1.1 List the OWASP Top 10 (2021).

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery (SSRF)

Q1.2 Why is OWASP Top 10 important?

It helps developers focus on the most common and critical web application security risks.

Q1.3 How often is OWASP Top 10 updated?

It is generally updated every 3–4 years to reflect new and emerging threats.

Q2. What are Secure Coding Practices?

Secure coding practices are guidelines developers follow to prevent security vulnerabilities in code, such as validating inputs, using prepared statements, and encrypting sensitive data.

Q2.1 Why is input validation important?

It prevents attackers from injecting malicious data into the system.

Q2.2 What is the least privilege principle?

Giving users or systems the minimum permissions needed to perform their tasks.

Q2.3 What is code review?

It is the process of examining code for security, quality, and logic issues before deployment.

Q3. What is SQL Injection?

SQL Injection is a web security vulnerability where an attacker can insert or manipulate SQL queries to access or modify database information.

Q3.1 How can SQL Injection be prevented?

Use prepared statements or parameterized queries instead of string concatenation.

Q3.2 Give an example of SQL Injection.

SELECT * FROM users WHERE username = ' ' OR '1'='1' -- ';

Q3.3 Which OWASP category does SQL Injection fall under?

It falls under “Injection” in OWASP Top 10.

Q4. What is Cross-Site Scripting (XSS)?

XSS allows attackers to inject malicious scripts into trusted websites, affecting users who visit the site.

Q4.1 What are the types of XSS?

1. Stored XSS

2. Reflected XSS

3. DOM-based XSS

Q4.2 How to prevent XSS?

Use output encoding, HTML escaping, and input validation.

Q4.3 Which OWASP category includes XSS?

“Injection”.

Q5. What is CSRF (Cross-Site Request Forgery)?

CSRF tricks users into performing actions on a web application where they are already authenticated.

Q5.1 How can CSRF be prevented?

Use anti-CSRF tokens and verify the origin or referer header.

Q5.2 What is a CSRF token?

It’s a unique, unpredictable value sent with each request to ensure the request is from a valid user.

Q5.3 Give an example of a CSRF attack.

A hidden form in a malicious site that triggers a bank transfer when the victim is logged in.

Q6. What is the difference between Authentication and Authorization?

- **Authentication:** Confirms user identity (who you are).
- **Authorization:** Determines what actions or data the user is allowed to access.

Q6.1 Example of authentication method?

Login with username and password.

Q6.2 Example of authorization check?

Allowing only admin users to delete records.

Q6.3 Can a user be authorized without authentication?

No, authentication must happen first.

Q7. What is Hashing?

Hashing converts data into a fixed-length string using algorithms like SHA-256. It’s one-way and cannot be reversed.

Q7.1 What is Salting?

Adding a random value (salt) to data before hashing to prevent rainbow table attacks.

Q7.2 Name common hashing algorithms.

MD5, SHA-1, SHA-256, SHA-512.

Q7.3 Why shouldn’t passwords be stored in plain text?

Because attackers can easily read them if the database is compromised.

Q8. What is Encryption?

Encryption converts plaintext into ciphertext to protect data confidentiality.

Q8.1 What is the difference between AES and RSA?

AES is symmetric (same key for encryption/decryption), RSA is asymmetric (public and private keys).

Q8.2 What is decryption?

The process of converting ciphertext back to plaintext.

Q8.3 Why is encryption important?

It protects data from being read by unauthorized users.

Q9. What is HTTPS?

HTTPS (Hypertext Transfer Protocol Secure) encrypts data between client and server using SSL/TLS for secure communication.

Q9.1 What port does HTTPS use?

Port 443.

Q9.2 What is the main benefit of HTTPS?

It ensures confidentiality, integrity, and authenticity of data in transit.

Q9.3 What protocol does HTTPS rely on for encryption?

TLS (Transport Layer Security).

Q10. What is Session Management?

Session management maintains a user's state between HTTP requests.

Q10.1 How is a session usually identified?

Through a session ID stored in cookies.

Q10.2 How can sessions be hijacked?

Through stolen cookies or unencrypted connections.

Q10.3 How to secure sessions?

Use secure, HttpOnly cookies and regenerate session IDs after login.

Q11. What is Input Validation?

Input validation ensures that data entered by users is correct, expected, and safe before processing.

Q11.1 What are the types of validation?

Client-side and server-side validation.

Q11.2 What can happen if inputs are not validated?

It can lead to attacks like SQL injection or XSS.

Q11.3 Example of input validation check?

Ensuring an email field contains a valid email format.

Q12. What are Prepared Statements?

Prepared statements are precompiled SQL queries that safely bind user inputs, preventing SQL injection.

Q12.1 Which languages support prepared statements?

Java (JDBC), Python (SQLite, MySQL), PHP, etc.

Q12.2 Why are they more secure?

Because parameters are treated as data, not executable code.

Q12.3 Example in Python?

```
cursor.execute("SELECT * FROM users WHERE id = ?", (user_id,))
```

Q13. What is Password Hashing?

It's the process of converting a password into a hashed value before storing it in a database.

Q13.1 Why is hashing used instead of encryption for passwords?

Because passwords don't need to be decrypted again.

Q13.2 What library is commonly used in Python for password hashing?

bcrypt or hashlib.

Q13.3 What happens if the same password is hashed twice?

It produces the same hash unless salt is added.

Q14. What is TLS (Transport Layer Security)?

TLS is a protocol that ensures privacy and data integrity between applications communicating over a network.

Q14.1 What was TLS previously called?

SSL (Secure Sockets Layer).

Q14.2 What does TLS protect against?

Eavesdropping and tampering.

Q14.3 Which layer of OSI model does TLS work on?

It operates between the transport and application layers.

Q15. What is Security Misconfiguration?

It occurs when security settings are not properly defined or left at default, making systems vulnerable.

Q15.1 Give an example.

Using default admin credentials like “admin/admin”.

Q15.2 How to prevent it?

Regular security audits and using secure defaults.

Q15.3 Which OWASP category is it?

“Security Misconfiguration”.

LAB-BASED QUESTIONS

Q16. How did you implement input validation in your project?

I validated user inputs both on the client and server side using regex and built-in validation functions to ensure safe data entry.

Q16.1 What happens if validation fails?

The user gets an error message, and the input is rejected.

Q16.2 Why do we need server-side validation even if we have client-side?

Because client-side validation can be bypassed.

Q16.3 Example of validation in Python?

```
if not re.match(r'^[a-zA-Z0-9]+$', username):
    return "Invalid username"
```

Q17. How did you prevent SQL Injection in your lab code?

I used prepared statements and parameterized queries instead of directly concatenating user input into SQL queries.

Q17.1 Why not use string concatenation in SQL?

Because it allows attackers to inject malicious SQL.

Q17.2 Which Python library supports prepared statements?

sqlite3, MySQL connector, psycopg2.

Q17.3 What does ? or %s represent in prepared statements?

It acts as a placeholder for parameters.

Q18. How did you implement password hashing in your project?

I used the bcrypt library to hash and salt user passwords before storing them in the database.

Q18.1 Why add salt before hashing?

To make each hash unique, even for the same passwords.

Q18.2 How to verify password during login?

By comparing the entered password hash with the stored hash.

Q18.3 Can you decrypt a hash?

No, hashing is one-way.

Q19. What is TLS configuration used for in secure apps?

TLS ensures secure data transmission between client and server using encryption.

Q19.1 What certificates are used in TLS?

SSL/TLS digital certificates issued by Certificate Authorities (CAs).

Q19.2 How can you verify if a site uses TLS?

By checking for “https://” and a padlock symbol in the browser.

Q19.3 What version of TLS is considered secure?

TLS 1.2 and TLS 1.3.

Q20. What is the purpose of using HTTPS in your project?

HTTPS secures communication and protects data from eavesdropping or tampering.

Q20.1 How is HTTPS different from HTTP?

HTTPS uses TLS encryption, HTTP doesn’t.

Q20.2 Can you host an HTTPS site without a certificate?

No, a digital certificate is required.

Q20.3 What happens if an SSL certificate expires?

The browser shows a “Not Secure” warning.

Q21. What is Input Sanitization?

Input sanitization means cleaning user input by removing or encoding dangerous characters before processing it.

Q21.1 How is it different from validation?

Validation checks *what* data is allowed; sanitization cleans *how* data is represented.

Q21.2 Example of sanitization?

Converting < and > into &lt; and &gt; to prevent XSS.

Q21.3 Should we do both validation and sanitization?

Yes — both together provide stronger protection.

Q22. What is Secure Error Handling?

Secure error handling prevents sensitive information like stack traces or database details from being shown to users.

Q22.1 Why should detailed errors be avoided?

They may reveal system structure or vulnerabilities to attackers.

Q22.2 What should users see instead?

Generic and user-friendly error messages.

Q22.3 Where should detailed logs be stored?

In secure server-side log files, not visible to users.

Q23. How do you secure file uploads in a web app?

Validate file type and size, rename the file, and store it outside the web root to prevent execution.

Q23.1 What kind of attacks are common in file uploads?

Malware uploads or script execution (e.g., .php shell).

Q23.2 How can you validate file types?

Check MIME type and file extension.

Q23.3 Should uploaded files ever be executed?

No, they should only be stored or processed safely.

Q24. What are Cookies used for in web security?

Cookies store session IDs or user data between requests.

Q24.1 What is the HttpOnly flag?

It prevents client-side scripts (like JavaScript) from accessing cookies.

Q24.2 What does the Secure flag do?

Ensures cookies are sent only over HTTPS.

Q24.3 What is the SameSite attribute?

It prevents cookies from being sent with cross-site requests (helps prevent CSRF).

Q25. What is JWT (JSON Web Token)?

JWT is a compact, secure way to transmit user authentication data between client and server using digitally signed tokens.

Q25.1 What are the 3 parts of a JWT?

Header, Payload, Signature.

Q25.2 Which algorithm is often used to sign JWTs?

HMAC SHA256 or RSA.

Q25.3 Where are JWTs usually stored?

In browser localStorage or cookies (preferably HttpOnly cookies).

Q26. What is API Security?

API security ensures that only authorized clients can access and modify data through an application's APIs.

Q26.1 What is an API key?

A unique identifier used to authenticate an API client.

Q26.2 What is rate limiting?

Restricting the number of API calls per user to prevent abuse.

Q26.3 Should APIs use HTTPS?

Always — to protect data in transit.

Q27. What is the Principle of Least Privilege?

It means giving users or systems only the minimum permissions required to perform their tasks.

Q27.1 Example of least privilege in databases?

Giving read-only access to reporting users.

Q27.2 Why is it important?

It limits the impact of a compromised account or bug.

Q27.3 How can developers apply it in code?

Use roles and fine-grained access control.

Q28. What is Dependency Management and why is it important for security?

It’s the process of keeping external libraries and frameworks up to date to avoid vulnerabilities.

Q28.1 How can you manage dependencies in Python?

Using requirements.txt and virtual environments.

Q28.2 Why is it risky to use outdated libraries?

They may contain known vulnerabilities.

Q28.3 How can you check for vulnerable dependencies?

Use tools like pip-audit or npm audit.

EXTENDED VIVA EXPLANATIONS (TYPES + OWASP DETAILS)

Q1. Explain the types of XSS (Cross-Site Scripting).

1. **Stored XSS:**
   The malicious script is permanently stored on the server (e.g., in a database or comment section). Every user who views that page gets affected.
   Example: A user posts a comment containing <script>alert('Hacked!')</script> which runs whenever others view it.
2. **Reflected XSS:**
   The malicious code is part of the request and reflected in the response, often through a URL parameter.
   Example: A crafted link that includes <script> in the query string.
3. **DOM-Based XSS:**
   The attack happens inside the browser when JavaScript modifies the DOM (Document Object Model) unsafely.
   Example: Client-side JS reads URL data and injects it into the page without sanitization.

Q2. Explain the types of SQL Injection.

1. **Classic (In-band) SQLi:**
   Attacker uses the same communication channel to inject and retrieve data (e.g., ' OR '1'='1').
2. **Blind SQLi:**
   No direct output — attacker infers information by observing server behavior (e.g., true/false responses).
3. **Out-of-band SQLi:**
   Data is sent to an external attacker-controlled server using DNS or HTTP requests.

Q3. Explain the types of CSRF attacks.

1. **Stored CSRF:**
   A malicious link or script is stored in the website and automatically executes when a valid user visits.
2. **Reflected CSRF:**
   The attacker tricks the victim into clicking a crafted URL that performs unwanted actions.
3. **Login CSRF:**
   Attacker forces the victim to log into an account controlled by the attacker.

Q4. Explain the types of Cryptography.

1. **Symmetric Encryption:**
   Same key used for encryption and decryption (e.g., AES).
   Fast but key sharing must be secure.
2. **Asymmetric Encryption:**
   Uses a public and private key pair (e.g., RSA).
   Secure for communication but slower.
3. **Hashing:**
   Converts data into a fixed-length hash (one-way). Used for passwords, not reversible.

Q5. Explain the types of Hashing Algorithms.

1. **MD5:** 128-bit hash, fast but insecure (collisions possible).
2. **SHA-1:** 160-bit hash, more secure than MD5 but now outdated.
3. **SHA-256 / SHA-512:** Modern, secure hash algorithms widely used for password protection.
4. **bcrypt / Argon2:** Adaptive hashing — includes salting and work factor to slow brute-force attacks.

Q6. Explain the types of Authentication methods.

1. **Password-based Authentication:** Username & password combo.
2. **Multi-Factor Authentication (MFA):** Combines password + OTP, fingerprint, or device.
3. **Token-based Authentication:** Uses tokens like JWT for session management.
4. **Biometric Authentication:** Uses fingerprints, facial recognition, or iris scan.

Q7. Explain the types of Authorization.

1. **Role-Based Access Control (RBAC):** Access based on roles (e.g., Admin, User).
2. **Attribute-Based Access Control (ABAC):** Access based on attributes like location, time, or device.
3. **Rule-Based Access Control:** Access based on specific system rules.

Q8. Explain the types of Encryption Algorithms.

1. **AES (Advanced Encryption Standard):** Symmetric key algorithm, secure and fast.
2. **RSA:** Asymmetric encryption using key pairs, commonly used for secure communication.
3. **DES/3DES:** Older symmetric encryption; less secure and replaced by AES.
4. **ECC (Elliptic Curve Cryptography):** Efficient and secure asymmetric encryption used in modern systems.

Q9. Explain the OWASP Top 10 (with simple explanations).

1. **Broken Access Control:**
   Users can access data or actions they shouldn't (e.g., normal user viewing admin pages).
2. **Cryptographic Failures:**
   Sensitive data not properly encrypted or exposed (e.g., using weak keys or plaintext storage).
3. **Injection:**
   Attackers insert malicious input (SQL, OS commands, etc.) to alter program behavior.
4. **Insecure Design:**
   Security not considered during system design (e.g., no input validation or misuse of logic).
5. **Security Misconfiguration:**
   Insecure default settings, open cloud storage, or unnecessary features enabled.
6. **Vulnerable and Outdated Components:**
   Using old libraries or frameworks with known security flaws.
7. **Identification and Authentication Failures:**
   Weak login systems, poor session management, or missing MFA.
8. **Software and Data Integrity Failures:**
   Unverified updates or dependencies leading to tampered software.
9. **Security Logging and Monitoring Failures:**
   Lack of proper logging allows attackers to hide their actions.
10. **Server-Side Request Forgery (SSRF):**
    Application fetches remote resources without validating user-supplied URLs.

Q10. Explain the types of Security Misconfiguration.

1. **Default Credentials:** Keeping “admin/admin” or “root/root.”
2. **Unnecessary Services Enabled:** Extra ports or debug mode open.
3. **Unpatched Systems:** Missing security updates.
4. **Verbose Error Messages:** Revealing sensitive details to attackers.

Q11. Explain the types of Session Attacks.

1. **Session Hijacking:** Stealing a valid user’s session ID to impersonate them.
2. **Session Fixation:** Attacker sets a known session ID and tricks the victim to use it.

3. **Session Replay:** Reusing old session tokens to gain access.

Q12. Explain the types of Input Validation.

1. **Whitelist Validation:** Accept only known safe input values.
2. **Blacklist Validation:** Reject known bad patterns (less secure).
3. **Server-Side Validation:** Performed on the backend, cannot be bypassed.
4. **Client-Side Validation:** Done using JavaScript for quick feedback (not sufficient alone).

Q13. Explain the types of Cookies (based on flags).

1. **HttpOnly Cookie:** Inaccessible by JavaScript — prevents XSS theft.
2. **Secure Cookie:** Sent only over HTTPS — prevents exposure over HTTP.
3. **SameSite Cookie:** Restricts cross-site sending — prevents CSRF.

Q14. Explain the types of API Authentication.

1. **API Keys:** Static token given to client for each request.
2. **OAuth 2.0:** Secure delegated access using access tokens.
3. **JWT Tokens:** Self-contained tokens signed by the server.
4. **Mutual TLS:** Both client and server verify each other via certificates.

Q15. Explain the types of TLS certificates.

1. **Domain Validation (DV):** Verifies domain ownership only.
2. **Organization Validation (OV):** Verifies business identity + domain.
3. **Extended Validation (EV):** Highest trust level; displays company name in browser bar.

Q16. Explain the types of Vulnerability Scanning.

1. **Static Application Security Testing (SAST):** Scans source code without running it.
2. **Dynamic Application Security Testing (DAST):** Tests the running application for issues.
3. **Dependency Scanning:** Checks third-party libraries for known vulnerabilities.

Q17. Explain the types of Logging in Secure Applications.

1. **Access Logs:** Track user access to resources.
2. **Error Logs:** Capture exceptions or failures.
3. **Audit Logs:** Record security-relevant events (like login attempts).

Q18. Explain the types of Attacks prevented by HTTPS.

1. **Man-in-the-Middle (MITM):** Prevents interception and tampering.

2. **Data Theft:** Prevents eavesdropping on credentials.
3. **Session Hijacking:** Encrypts cookies and tokens in transit.

Q19. Explain the types of Authentication Attacks.

1. **Brute Force Attack:** Tries all possible passwords.
2. **Credential Stuffing:** Uses leaked username-password pairs.
3. **Phishing:** Tricks users into revealing credentials.
4. **Password Spraying:** Tries common passwords across many accounts.

Q20. Explain the types of Cryptographic Attacks.

1. **Brute Force:** Trying all keys.
2. **Dictionary Attack:** Using a list of common words/passwords.
3. **Rainbow Table Attack:** Precomputed hashes used to reverse weak hashes.
4. **Man-in-the-Middle (MITM):** Intercepting keys or data during transmission.

FINAL ADVANCED & EDGE CASE QUESTIONS (for Complete Coverage)

Q1. What is Clickjacking?

Clickjacking is when an attacker tricks a user into clicking on something different from what they think — usually by overlaying invisible frames.

Example: A fake “Play” button that actually clicks “Delete Account.”

Q1.1 How to prevent Clickjacking?

Use the X-Frame-Options or Content-Security-Policy: frame-ancestors header.

Q1.2 What are X-Frame-Options?

It prevents your web page from being displayed inside an iframe.

Q1.3 Which OWASP category includes Clickjacking?

It’s part of “Insecure Design” or “Security Misconfiguration.”

Q2. What is Content Security Policy (CSP)?

CSP is an HTTP header that helps prevent attacks like XSS by controlling what resources (scripts, images, etc.) can load on a page.

Q2.1 Example of CSP header?

Content-Security-Policy: default-src 'self'; script-src 'self';

Q2.2 Can CSP block inline scripts?

Yes, it blocks inline JavaScript by default.

Q2.3 What happens if CSP is misconfigured?

It may block legitimate resources or still allow unsafe ones.

Q3. What is a Security Header?

Security headers are HTTP response headers that add extra layers of protection for browsers and web apps.

Q3.1 Name some important security headers.

- Content-Security-Policy (CSP)
- Strict-Transport-Security (HSTS)
- X-Frame-Options
- X-Content-Type-Options
- Referrer-Policy

Q3.2 What does HSTS do?

Forces browsers to connect using HTTPS only.

Q3.3 What does X-Content-Type-Options: nosniff do?

Prevents browsers from interpreting files as a different MIME type.

Q4. What is a Man-in-the-Middle (MITM) attack?

In MITM, an attacker secretly intercepts and possibly alters communication between two parties.

Q4.1 How to prevent MITM attacks?

Use HTTPS/TLS and certificate pinning.

Q4.2 What is certificate pinning?

It ensures the client only trusts a specific known certificate or public key.

Q4.3 Can VPNs protect from MITM?

Yes, because they encrypt all network traffic.

Q5. What is a Brute-Force Attack and how do you prevent it?

A brute-force attack tries every possible password combination to gain unauthorized access.

Q5.1 How can it be prevented?

Use account lockout, CAPTCHA, and rate limiting.

Q5.2 What is password salting's role here?

It prevents precomputed hash attacks like rainbow tables.

Q5.3 Why should we avoid short passwords?

Shorter passwords reduce the number of possible combinations, making attacks faster.

Q6. What is Logging and Monitoring in security?

It's the process of recording application and system events to detect and respond to security incidents.

Q6.1 Why is it important?

It helps detect attacks and trace what happened.

Q6.2 What should not be logged?

Sensitive data like passwords or credit card numbers.

Q6.3 Which OWASP category covers this?

"Security Logging and Monitoring Failures."

Q7. What is a Security Token?

A token is a piece of data representing user authentication or session identity.

Q7.1 Example of a security token?

JWT (JSON Web Token) or CSRF token.

Q7.2 How is a JWT structured?

Header + Payload + Signature (separated by dots).

Q7.3 Why should JWTs expire?

To prevent reuse if stolen.

Q8. What is Secure File Handling?

It involves validating, renaming, and storing user-uploaded files safely to prevent malicious uploads.

Q8.1 What is path traversal?

Attackers try to access restricted files using ../ in paths.

Q8.2 How to prevent path traversal?

Use os.path.join() safely and validate file paths.

Q8.3 Should uploaded files be executed?

Never — they should only be stored or processed as data.

Q9. What is Rate Limiting?

Rate limiting controls how many requests a user or IP can make in a certain time.

Q9.1 Why is it used?

To prevent brute-force attacks and API abuse.

Q9.2 Example of rate limiting?

“Max 5 login attempts per minute per user.”

Q9.3 What happens if it’s too strict?

It can affect user experience or block legitimate requests.

Q10. What is a Secure Development Lifecycle (SDL)?

SDL integrates security at every stage of the software development process — from design to deployment.

Q10.1 Why is SDL important?

It reduces vulnerabilities early when they’re cheaper to fix.

Q10.2 Name SDL stages.

Requirements → Design → Implementation → Testing → Deployment → Maintenance.

Q10.3 What tool can help automate security checks?

Static code analysis tools (like SonarQube, Bandit, etc.).

Q11. What is a Zero-Day Vulnerability?

It’s a security flaw unknown to the vendor and unpatched, giving attackers an advantage.

Q11.1 How can developers reduce zero-day risk?

Keep software updated and use intrusion detection.

Q11.2 What does “exploit” mean?

Code or method that takes advantage of a vulnerability.

Q11.3 What’s the best defense against zero-day attacks?

Defense-in-depth (multiple layers of security).

Q12. What is the Least Common Mechanism principle?

It means minimizing shared resources among users to reduce cross-user interference and risk.

Q12.1 Example of violating this principle?

Sharing the same system process for multiple users.

Q12.2 Why is it important in web apps?

Shared cookies or sessions can lead to data leaks.

Q12.3 How can developers follow it?

Use isolated environments per user or microservices.

Q13. What is the difference between Static and Dynamic Code Analysis?

- **Static Analysis:** Checks source code for vulnerabilities before running.
- **Dynamic Analysis:** Tests the application while it’s running.

Q13.1 Give example tools.

Static: Bandit, SonarQube; Dynamic: OWASP ZAP, Burp Suite.

Q13.2 Which is faster?

Static analysis is faster but may miss runtime issues.

Q13.3 Which is more realistic?

Dynamic, since it tests actual execution.

Q14. What is Security by Obscurity?

Hiding system details (like directory names or code) to make attacks harder — but not a real defense by itself.

Q14.1 Is it good practice?

No — it should never be the only protection.

Q14.2 Example?

Renaming “admin.php” to “hidden123.php” — not true security.

Q14.3 What’s better than obscurity?

Proper authentication, encryption, and access control.

Q15. What is Penetration Testing?

Pen testing is the simulated attack on a system to find vulnerabilities before real attackers do.

Q15.1 What are the types of pen testing?

1. Black Box — no internal knowledge
2. White Box — full internal access
3. Gray Box — partial knowledge

Q15.2 What tools are used?

Burp Suite, Metasploit, OWASP ZAP.

Q15.3 When should it be done?

Before deployment or major updates.

SECURE APPLICATION DEVELOPMENT LAB – TECHNICAL TERM EXPLANATIONS

1. Secure Coding

Writing code in a way that prevents vulnerabilities and attacks.

Example: Validating inputs, encrypting passwords, using prepared statements.

2. Input Validation

Checking user input to make sure it’s safe, expected, and properly formatted.

Prevents attacks like SQL Injection and XSS.

3. Input Sanitization

Cleaning user input by removing or encoding unsafe characters before using it.

Used mainly to stop XSS or HTML injection.

4. Prepared Statement

A precompiled SQL query where user inputs are passed safely as parameters.

Prevents SQL Injection by separating data from SQL code.

5. Parameterized Query

Similar to prepared statements — a query that uses placeholders (?, %s) for data instead of string concatenation.

6. SQL Injection (SQLi)

An attack that manipulates SQL queries by inserting malicious input.

Example: ' OR '1'='1'.

7. Cross-Site Scripting (XSS)

Injection of malicious scripts (usually JavaScript) into trusted websites.

Types: Stored, Reflected, DOM-Based.

8. Cross-Site Request Forgery (CSRF)

An attack that tricks a logged-in user into making unwanted actions (like changing a password) without consent.

9. OWASP

(Open Web Application Security Project) — a global community that lists common web app security risks (OWASP Top 10).

10. OWASP Top 10

The ten most critical web application security risks — like Broken Access Control, Injection, XSS, etc.

11. Authentication

The process of verifying who a user is (e.g., login using username and password).

12. Authorization

Determining what a user is allowed to do (e.g., admin can delete data, user cannot).

13. Session Management

Maintaining a user's session (logged-in state) across multiple requests.

Uses session IDs or cookies.

14. Cookies

Small data pieces stored in the browser to maintain user sessions or preferences.

Important flags: HttpOnly, Secure, SameSite.

15. JWT (JSON Web Token)

A compact, signed token (Header.Payload.Signature) used for authentication between client and server.

16. HTTPS

(Hypertext Transfer Protocol Secure) – encrypts communication between client and server using SSL/TLS.

17. SSL/TLS

Protocols for secure communication over the internet.

TLS (Transport Layer Security) is the modern version of SSL.

18. Encryption

Converting plain data into unreadable form (ciphertext).

Used to protect sensitive data in storage or transmission.

19. AES (Advanced Encryption Standard)

A symmetric encryption algorithm (same key for encryption and decryption).

Fast and secure.

20. RSA

An asymmetric encryption algorithm that uses a *public key* for encryption and a *private key* for decryption.

21. Hashing

A one-way process that converts data into a fixed-length hash value.

Used to securely store passwords (e.g., SHA-256).

22. Salting

Adding a random string to passwords before hashing to make hashes unique and prevent rainbow table attacks.

23. Password Hashing

Storing passwords as hashed (and salted) values instead of plain text.

Example tools: bcrypt, Argon2, hashlib.

24. Cryptography

The science of securing communication by encoding information (includes hashing, encryption, and digital signatures).

25. Digital Signature

A cryptographic method to verify authenticity and integrity of a message or document using a private key.

26. Man-in-the-Middle (MITM) Attack

When an attacker intercepts communication between two parties to read or alter it.

Prevented by HTTPS/TLS.

27. Security Misconfiguration

Leaving systems with default or insecure settings (like default passwords or debug mode enabled).

28. Vulnerable and Outdated Components

Using old libraries or frameworks that have known vulnerabilities.

29. Logging

Recording application events, user activities, and errors for auditing and troubleshooting.

30. Monitoring

Actively observing system logs and behavior to detect attacks or failures early.

31. Brute Force Attack

Repeatedly trying many passwords or keys until the correct one is found.

32. Rate Limiting

Restricting how many times a user or IP can make a request (used to stop brute-force attacks).

33. Firewall

A system that filters incoming and outgoing network traffic based on security rules.

34. IDS / IPS

- **IDS (Intrusion Detection System):** Detects suspicious activity.
- **IPS (Intrusion Prevention System):** Detects and blocks it.

35. Security Headers

HTTP headers that enhance security, like:

- Strict-Transport-Security (forces HTTPS)
- Content-Security-Policy (prevents XSS)
- X-Frame-Options (prevents clickjacking)

36. Content Security Policy (CSP)

An HTTP header that controls which resources (scripts, styles, etc.) can load on a web page.

37. Clickjacking

Tricking users into clicking something different from what they see — prevented by X-Frame-Options and CSP.

38. Same-Origin Policy

A browser security rule that restricts how documents from different origins can interact.

39. CORS (Cross-Origin Resource Sharing)

A mechanism that allows or restricts resource sharing between different domains safely.

40. Security Audit

A systematic review of an application to find security weaknesses.

41. Principle of Least Privilege

Give users or systems only the minimum access needed to perform their tasks.

42. Defense in Depth

Layering multiple security measures so if one fails, others still protect the system.

43. Static Application Security Testing (SAST)

Analyzing source code without running it to detect vulnerabilities.

44. Dynamic Application Security Testing (DAST)

Testing a running application to find real-time security issues.

45. Penetration Testing

Simulated attack by ethical hackers to find vulnerabilities before real attackers do.

46. Secure Development Lifecycle (SDL)

Integrating security into every phase of software development — from design to deployment.

47. Dependency Management

Keeping external libraries updated and verified to avoid known security flaws.

48. Zero-Day Vulnerability

A flaw unknown to the vendor — no patch yet available.

49. Token

A piece of data that represents identity or access (e.g., session token, JWT, CSRF token).

50. CSRF Token

A unique random value used to confirm that a form submission came from the legitimate user, preventing CSRF.

51. Session ID

A unique number assigned to a user’s session, stored in a cookie or URL.

52. Public Key

Used to encrypt data or verify signatures.

Shared openly.

53. Private Key

Used to decrypt data or create digital signatures.

Kept secret.

54. Certificate Authority (CA)

An organization that issues digital certificates to verify identity on the internet.

55. Certificate Pinning

Associating a server with a specific certificate or public key to prevent fake certificates (used to stop MITM).

56. API Security

Ensuring APIs are accessed safely through authentication, rate limits, and HTTPS.

57. Environment Variables

Used to store sensitive configuration values (like passwords, API keys) securely outside the source code.

58. Secure File Upload

Verifying file type, renaming, and storing files safely to avoid uploading malware or scripts.

59. Path Traversal Attack

Using special characters like ../ to access files outside the allowed directory.

60. Dependency Vulnerability Scanning

Automatically checking third-party libraries for known security flaws.

61. Data Integrity

Ensuring data isn't modified or corrupted during transmission or storage.

62. Data Confidentiality

Ensuring that data is accessible only to authorized users.

63. Data Availability

Ensuring that information and systems are available when needed.

64. Defense Mechanism

Any safeguard that prevents or reduces security risks — like validation, encryption, or authentication.

65. Secure Configuration

Setting systems and software to the most secure state — disable unnecessary features, enforce strong passwords, update regularly.

Excellent 🔥 — that's a *very smart* catch. Examiners often move from **"secure coding"** to **REST APIs and communication flow** because they want to check if you understand how **client and server interact securely**.

Below is your **exam-ready explanation set**, written exactly in the same style as before — **simple, clear, and viva-speakable** 💬

REST API & CLIENT–SERVER COMMUNICATION (VIVA ANSWERS)

Q1. What is a REST API?

REST stands for **Representational State Transfer**.

It's an **architecture style** used to build web services where a **client communicates with a server** using **HTTP methods** like GET, POST, PUT, and DELETE.

It is **stateless**, meaning every request from the client must contain all the information the server needs to process it.

Q1.1 What is meant by "stateless" in REST?

Stateless means the **server doesn't store client session data** between requests.

Each request is **independent** and must carry authentication or data needed for that request.

Q1.2 What is an endpoint in REST API?

An **endpoint** is a specific URL or route where the client can access or send data.

Example:

https://api.example.com/users → endpoint for user-related data.

Q1.3 What is the format of data sent in REST APIs?

Usually JSON (JavaScript Object Notation) is used, but XML can also be used.

Example JSON data:

```
{
  "username": "vanshita",
  "password": "12345"
}
```

Q2. How does a client send a message or request to the server?

A client (like a browser or mobile app) sends an **HTTP request** to the server using a method such as **GET**, **POST**, **PUT**, or **DELETE**.

The server processes it and returns an **HTTP response**.

Q2.1 Explain the basic structure of an HTTP request.

An HTTP request has **three parts**:

1. **Request Line:** Method, URL, and HTTP version (e.g., POST /login HTTP/1.1)
2. **Headers:** Additional info like Content-Type, Authorization
3. **Body:** Actual data sent (like JSON or form data)

Example:

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "username": "vanshita",
  "password": "mypassword"
}
```

Q2.2 What are HTTP methods used in REST APIs?

1. **GET** → Retrieve data from the server
2. **POST** → Send new data to the server
3. **PUT** → Update existing data
4. **DELETE** → Remove data
5. **PATCH** → Partially update data

Q2.3 How does the server respond to the client?

The server sends back an **HTTP response** which includes:

- **Status code** (e.g., 200 OK, 404 Not Found, 500 Server Error)
- **Headers** (like Content-Type)
- **Body** (data, usually in JSON)

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Login successful",
  "token": "abc123xyz"
}
```

Q3. What are HTTP Status Codes?

They tell the client whether the request was successful or failed.

Common ones:

- 200 → OK (success)
- 201 → Created
- 400 → Bad Request
- 401 → Unauthorized
- 403 → Forbidden
- 404 → Not Found
- 500 → Internal Server Error

Q3.1 What does “400 Bad Request” mean?

It means the client sent invalid or incomplete data to the server.

Q3.2 What does “401 Unauthorized” mean?

It means the request lacks valid authentication credentials (e.g., missing or wrong token).

Q3.3 What does “500 Internal Server Error” mean?

It means something went wrong on the server side while processing the request.

Q4. What is an API Request Body?

It’s the **data part** of the request sent from the client to the server, especially in POST or PUT methods.

Example (JSON body):

```
{
  "name": "Vanshita",
  "email": "vanshita@example.com"
}
```

Q4.1 What does the “Content-Type” header mean?

It tells the server what kind of data is being sent.

Example:

Content-Type: application/json means the body is in JSON format.

Q4.2 What is a Request Header?

It carries metadata like content type, API key, or authentication token with the request.

Q4.3 What is a Response Header?

It gives information about the server or response, like data type or caching details.

Q5. What is JSON and why is it used in REST APIs?

JSON (JavaScript Object Notation) is a lightweight data format used to exchange data between client and server.

It's human-readable and easy for both machines and humans to process.

Q5.1 Difference between JSON and XML?

- JSON is lightweight and easy to read
- XML is heavier and uses opening/closing tags
  Example:
  JSON → {"name": "Vanshita"}
  XML → <name>Vanshita</name>

Q5.2 What does "application/json" mean?

It's a MIME type that tells the server the request or response body contains JSON data.

Q5.3 Is JSON language-dependent?

No, it's language-independent — can be used in Python, Java, C#, etc.

Q6. How does a REST API maintain security?

By using:

1. **HTTPS** → Encrypts communication
2. **Authentication** → API keys, tokens, or JWT
3. **Authorization** → Role-based access
4. **Validation** → Checks data integrity
5. **Rate limiting** → Prevents abuse

Q6.1 What is an API key?

A unique code used to authenticate the client with the server in each request.

Q6.2 What is Token-based Authentication?

After login, the server gives a **token** (like JWT).

Client sends this token with every request to prove identity.

Q6.3 Example of sending a token in a request?

```
GET /user/profile HTTP/1.1
Host: example.com
Authorization: Bearer abc123xyz
```

Q7. What is REST vs SOAP?

Feature	REST	SOAP
Protocol	Uses HTTP	Uses XML over HTTP/SMTP
Format	JSON or XML	XML only
Speed	Fast	Slower
Simplicity	Simple	Complex
State	Stateless	Stateful or Stateless

Q7.1 Which is preferred for web apps?

REST — because it's lightweight and easy to use with JavaScript and Python.

Q7.2 Can REST APIs send files?

Yes — using multipart/form-data or binary formats like Base64.

Q7.3 What is RESTful Service?

A web service that follows REST principles — uses HTTP methods, statelessness, and resource-based URLs.

Q8. How does a REST API handle errors?

By sending **status codes** and a **JSON message** describing the error.

Example:

```
{
  "error": "Invalid input",
  "code": 400
```

}

Q8.1 Why are standard codes used?

So clients and servers can understand responses consistently.

Q8.2 What is error handling in APIs?

It means giving clear, structured messages when something fails.

Q8.3 Why is logging important in APIs?

It helps identify bugs, track usage, and detect attacks.

Q9. What is a REST Client?

It’s a tool or software that sends HTTP requests to test APIs.

Examples: Postman, Curl, or Python requests library.

Q9.1 What is Postman used for?

Postman is an API testing tool used to send requests (GET, POST, PUT, DELETE) and check responses.

Q9.2 Example of sending data using Python’s requests module?

```
import requests
data = {"name": "Vanshita"}
response = requests.post("https://api.example.com/users", json=data)
print(response.text)
```

Q9.3 What is the role of a REST API in full-stack development?

It acts as the **bridge between frontend and backend**, letting them exchange data securely and efficiently.