OWASP Top 10 Remediation steps:

1. Injection:
- Always use prepared statements or parameterized queries in your code.
- Validate and sanitize all user input before using it.

2. Broken Authentication:
- Use strong passwords and enable multi-factor authentication.
- Implement secure session management techniques.

3. Sensitive Data Exposure:
- Use strong encryption algorithms and salted passwords.
- Store sensitive data in secure locations, such as a dedicated hardware security module (HSM).

4. XML External Entities (XXE):
- Disable XML external entities.
- Use a different parser or library that is not vulnerable to XXE attacks.
- Validate and sanitize all user input.

5. Broken Access Control:
- Implement proper access controls and role-based access control (RBAC).
- Enforce access controls on both the client and server side.

6. Security Misconfiguration:
- Follow secure coding practices and keep software and servers up to date with the latest patches.
- Use tools like vulnerability scanners to identify potential vulnerabilities.

7. Cross-Site Scripting (XSS):
- Validate and sanitize all user input.
- Encode output properly.
- Implement Content Security Policy (CSP) to restrict the sources of content that can be loaded by a page.

8. Insecure Deserialization:
- Use safe deserialization methods or libraries.
- Validate and sanitize all user input.
- Implement input validation on the server side.

By following these remediation steps, web application developers can reduce the risk of vulnerabilities and improve the security of their applications.

XSS and its types Remediation

1. Reflected XSS:
In Reflected XSS, the attacker injects malicious code into a URL or form input that is immediately reflected back to the user's browser. This can trick the user into executing the code and revealing sensitive information.

Remediation:
- Validate and sanitize all user input before it is used.
- Encode output properly.
- Implement Content Security Policy (CSP) to restrict the sources of content that can be loaded by a page.

2. Stored XSS:
In Stored XSS, the attacker injects malicious code that is permanently stored on the server. This can affect all users who access the affected page, and the malicious code can continue to execute even after the attacker has left.

Remediation:
- Use a secure coding practice, such as input validation and sanitization.
- Implement a web application firewall (WAF) to detect and block attacks.
- Implement output encoding to prevent the execution of malicious code.

3. DOM-Based XSS:
In DOM-Based XSS, the attacker injects malicious code that is executed by the client-side script, rather than by the server. This can be difficult to detect and prevent, as the code may only affect a specific part of the page.

Remediation:
- Implement input validation and sanitization on the client-side as well as the server-side.
- Use a web application scanner to detect vulnerabilities.
- Use an updated browser that supports CSP.

By following these remediation steps and being aware of the different types of XSS attacks, web application developers can better protect their users and prevent the exploitation of XSS vulnerabilities.

Remediation of LFI vulnerability:

Local File Inclusion (LFI) is a type of vulnerability that allows attackers to include files on a server that are not intended to be accessed. This can allow an attacker to view sensitive information, such as passwords or configuration files.

Remediation:
- Use a secure coding practice, such as input validation and sanitization.
- Implement a web application firewall (WAF) to detect and block attacks.
- Avoid using user-supplied input to construct file paths.
- If user-supplied input is required, ensure that it is properly sanitized and validated.
- Implement a whitelist-based approach to input validation.
- Restrict file permissions to limit access to sensitive files.
- Use a chrooted environment or containerization to isolate the application.

By following these remediation steps, web application developers can better protect their users and prevent the exploitation of LFI vulnerabilities.