**1. SQL Injection (SQLi)**

SQL injection is a common attack where cybercriminals insert malicious SQL queries into web forms or URLs. This is possible when the application doesn’t properly validate or sanitize user input. For example, if someone enters harmful code into a search box, the app may end up executing commands that can manipulate the database.

**Risks and Impact:**

* **Confidentiality**: Attackers can gain access to sensitive data like customer information, credit card numbers, or even passwords stored in the database.
* **Integrity**: An attacker can alter, delete, or corrupt data within the database.
* **Availability**: If a malicious query crashes the database, it could disrupt service, causing downtime.
* **Legal & Financial**: Organizations could face significant fines for breaching data protection laws like GDPR or HIPAA. There may also be costly investigations, class-action lawsuits, and compensation payouts.
* **Reputation**: Customers lose trust when they learn their data wasn’t protected properly.

**How to Prevent It:**

* **Prepared Statements**: Always use parameterized queries to handle user input safely. This ensures the input is treated as data, not executable code.
* **Input Validation**: Only allow expected characters in user inputs (e.g., no special characters that could manipulate SQL queries).
* **Limit Database Permissions**: Grant the database the least privileges necessary. If attackers do get access, the damage will be limited.

**Risk Mitigation:**

* Regular security testing (penetration testing) to identify vulnerabilities.
* Implementing Web Application Firewalls (WAFs) to detect and block malicious traffic.

**2. Cross-Site Scripting (XSS)**

XSS occurs when an attacker injects malicious scripts into web pages viewed by other users. These scripts run in the context of the user's browser, potentially stealing cookies, session tokens, or other sensitive data.

**Risks and Impact:**

* **Confidentiality**: Sensitive data like login credentials or session tokens can be stolen and used for unauthorized access.
* **Integrity**: Attackers can manipulate content on the website, causing users to click on fake links or exposing them to malicious software.
* **Availability**: If users are scared away by security issues, the website could see a drop in traffic or usage.
* **Legal & Financial**: Exposed personal data can result in fines and legal action under privacy laws. Financial costs include remediation, legal fees, and potentially compensating affected users.
* **Reputation**: Customers and users may lose trust in a brand after learning it was vulnerable to XSS attacks.

**How to Prevent It:**

* **Content Security Policy (CSP)**: Use a CSP to restrict where scripts can be loaded from, reducing the risk of malicious script execution.
* **Output Encoding**: Always encode user inputs before displaying them on web pages, preventing scripts from executing.
* **Sanitize Input**: Clean any user-supplied data to remove harmful elements before processing or displaying it.

**Risk Mitigation:**

* Regularly audit code and perform security scans to identify XSS vulnerabilities.
* Educate users on security best practices, like avoiding suspicious links.

**3. Cross-Site Request Forgery (CSRF)**

In a CSRF attack, the attacker tricks an authenticated user into performing actions on a website without their knowledge. For example, clicking a malicious link could change account details or make unauthorized transactions on a banking site.

**Risks and Impact:**

* **Confidentiality**: Sensitive account details could be altered or compromised.
* **Integrity**: An attacker might change settings or perform actions that impact the system’s integrity (like transferring funds from an account).
* **Availability**: While CSRF doesn’t typically target availability, the harm done to users can result in them avoiding the site altogether.
* **Legal & Financial**: Unauthorized transactions could lead to legal action and financial losses. There may be regulatory fines if the breach involves personal financial data.
* **Reputation**: An organization may be seen as insecure, leading to a loss of customer trust.

**How to Prevent It:**

* **Anti-CSRF Tokens**: Use tokens in requests to ensure they are coming from a valid source and not an attacker.
* **SameSite Cookies**: Configure cookies with the SameSite attribute to restrict how they’re sent with cross-origin requests.
* **Session Management**: Regularly check session integrity and expire sessions after a period of inactivity.

**Risk Mitigation:**

* Educate users about phishing and the risks of clicking unknown links.
* Enforce two-factor authentication (2FA) to further secure user accounts.

**4. Broken Authentication and Session Management**

When a web application doesn’t securely manage user authentication, attackers can exploit weak or leaked credentials to impersonate users. This might happen if a session is not properly expired, or passwords are too easy to guess.

**Risks and Impact:**

* **Confidentiality**: Attackers may steal user credentials or access sensitive data without authorization.
* **Integrity**: Malicious users can alter data or execute unauthorized actions on the platform.
* **Availability**: If attackers lock legitimate users out of their accounts or gain full control, the availability of critical services could be jeopardized.
* **Legal & Financial**: Organizations could face legal challenges if the breach results in the exposure of private user information.
* **Reputation**: Trust is severely damaged when an attacker gains access to users' personal information, especially in sectors like banking or healthcare.

**How to Prevent It:**

* **Strong Passwords**: Enforce strong password requirements and educate users about password security.
* **Multi-Factor Authentication (MFA)**: Add an additional layer of security to prevent unauthorized access even if passwords are compromised.
* **Secure Session Management**: Implement secure session timeout, automatic logouts, and ensure that session cookies are marked as HttpOnly and Secure.

**Risk Mitigation:**

* Continuously monitor login activity and watch for unusual patterns.
* Implement account lockout mechanisms to prevent brute-force attacks.

**5. Insecure Deserialization**

Insecure deserialization occurs when an application deserializes untrusted data, allowing an attacker to execute arbitrary code or bypass security controls. This often happens when objects are improperly validated before being processed.

**Risks and Impact:**

* **Confidentiality**: Attackers may use deserialization to gain access to sensitive data by injecting malicious code.
* **Integrity**: An attacker can modify the behavior of the application or corrupt data.
* **Availability**: A successful attack could crash the application or disrupt services.
* **Legal & Financial**: Breaches involving exposed data could result in fines, and fixing the vulnerability can be costly.
* **Reputation**: Customers lose confidence in an organization that fails to secure their data.

**How to Prevent It:**

* **Avoid Serialization**: If possible, avoid deserialization altogether, or only use safe formats (like JSON) instead of vulnerable formats (like Java objects).
* **Data Validation**: Always validate and sanitize data before deserializing it.
* **Use Safe Libraries**: Ensure any deserialization libraries used are secure and up to date.

**Risk Mitigation:**

* Regular vulnerability scans to identify deserialization risks.
* Use application monitoring tools to detect unexpected behaviors or attacks.