# SQL Injection

SQL injection is a type of web application security vulnerability that occurs when an attacker can manipulate the SQL queries used by a web application to interact with its underlying database. By injecting malicious SQL code into the application's input fields or parameters, an attacker can gain unauthorized access to the database, modify data, or perform other malicious actions.

## **How SQL Injection Works:**

**User Input:** Web applications often take user input through forms, search boxes, or URLs to process data or perform actions.

**Lack of Input Validation:** If the web application doesn't properly validate or sanitize the user input before using it in an SQL query, an attacker can exploit this vulnerability.

**Injecting Malicious Code:** The attacker crafts input that includes SQL commands or fragments that alter the original query's behavior.

**SQL Execution:** When the manipulated input is processed by the web application, the malicious SQL code gets executed along with the legitimate query, potentially revealing sensitive information or compromising the database.

## **Types of SQL Injection:**

**Classic SQL Injection:** The attacker injects SQL code directly into input fields.

Blind SQL Injection: The attacker exploits the vulnerability indirectly by asking true or false questions to the database and interpreting the application's response.

Impact of SQL Injection:

**Unauthorized Access:** Attackers can gain access to sensitive data such as usernames, passwords, personal information, or financial records.

Data Manipulation: They can modify, delete, or insert data into the database, leading to data corruption or loss.

**Privilege Escalation:** If the database is connected with administrative privileges, attackers could gain full control over the database and the web application.

## **Preventing SQL Injection:**

**Parameterized Queries:** Use parameterized or prepared statements with placeholders for user inputs. This approach automatically escapes input, making it safe from SQL injection.

Stored Procedures: Utilize stored procedures instead of embedding SQL queries in the application code.

**Least Privilege Principle:** Configure the database to restrict user accounts' permissions to only the necessary operations and avoid granting excessive privileges.

Input Validation and Sanitization: Implement strong input validation and data sanitization to prevent malicious input.

**Web Application Firewall (WAF):** Employ a WAF to filter and block malicious SQL injection attempts.

**Regular Security Audits:** Regularly perform security audits, code reviews, and penetration tests to identify and fix vulnerabilities.

# 

# HTML Injection

HTML injection, also known as "Client-Side Injection" or "DOM-based XSS (Cross-Site Scripting)," is a web application security vulnerability that allows an attacker to inject and execute malicious HTML code within a web application's pages. Unlike traditional XSS, which targets server-side vulnerabilities, HTML injection focuses on manipulating the client-side (browser) behavior of a web application.

## How HTML Injection Works:

**User Input:** Similar to other web application vulnerabilities, HTML injection exploits the lack of proper input validation or sanitization of user-supplied data.

**Injecting Malicious HTML:** An attacker crafts input that includes HTML tags, attributes, or scripts to be executed by the browser.

**Execution:** When the user's input is processed by the web application and displayed on the page, the injected HTML code executes in the user's browser, leading to unintended behavior.

## Types of HTML Injection:

**Persistent HTML Injection:** The malicious HTML code is stored permanently in the application's database and displayed to all users who view the affected page.

**Non-Persistent (Reflected) HTML Injection:** The injected HTML code is reflected back to the user in the application's response. This type is often part of a social engineering attack where the attacker tricks the user into clicking on a malicious link that contains the payload.

## Impact of HTML Injection:

**Malicious Content Display:** Attackers can inject fake content, phishing forms, or misleading information, tricking users into divulging sensitive data.

**Cross-Site Scripting (XSS):** HTML injection can lead to the execution of malicious scripts, effectively turning it into a type of XSS attack.

## Preventing HTML Injection:

**Input Validation and Sanitization:** Validate and sanitize all user-supplied data, especially when rendering it on the page. Use appropriate escaping or encoding techniques to prevent HTML injection.

**Content Security Policy (CSP):** Implement CSP to restrict the sources from which the browser can load scripts, styles, and other resources. This can help prevent the execution of injected scripts.

**Use Safe HTML Rendering Libraries**: If your application requires rendering HTML from untrusted sources, use libraries that have built-in protection mechanisms against HTML injection, such as DOMPurify.

**Context-Aware Escaping:** Apply context-aware escaping to ensure that user input is rendered as plain text and not executed as HTML.

**Regular Security Audits:** Conduct regular security assessments and code reviews to identify and address vulnerabilities, including HTML injection.

# XSS

Cross-Site Scripting (XSS) is a common web application security vulnerability that allows attackers to inject and execute malicious scripts (usually JavaScript) into web pages viewed by other users. XSS attacks occur when a web application does not properly validate or sanitize user input before rendering it on the page, allowing attackers to inject code that gets executed in the victim's browser.

## Types of XSS attacks:

**Stored XSS (Persistent XSS):** The injected malicious script is permanently stored on the server (e.g., in a database) and gets displayed to every user who visits the vulnerable page. This type of attack is dangerous as it can affect multiple users and may persist over time.

**Reflected XSS (Non-Persistent XSS):** In this type of attack, the malicious script is included in the URL or user input, and the injected script is reflected back as part of the application's response. The user needs to be tricked into clicking on a crafted link to execute the script, making it more reliant on social engineering.

**DOM-based XSS (DOM XSS):** This variant occurs when the malicious script manipulates the Document Object Model (DOM) of a web page directly. The injected script is not sent to the server, making it challenging to detect using traditional server-side security measures.

## Impact of XSS Attacks:

**Data Theft:** Attackers can steal sensitive information, such as cookies, login credentials, or personal data, from the victim's browser.

**Session Hijacking:** XSS can be used to impersonate users and perform actions on their behalf.

Defacement: Attackers can modify the appearance of a website or deface it with malicious content.

**Phishing:** Malicious scripts can be used to create convincing phishing forms to trick users into disclosing their credentials.

## Preventing XSS Attacks:

**Input Validation and Sanitization:** Validate and sanitize all user-supplied data before displaying it on web pages to prevent malicious code injection.

**Content Security Policy (CSP):** Implement CSP to restrict the sources from which the browser can load scripts, styles, and other resources, reducing the risk of XSS attacks.

**Output Encoding:** Encode dynamic content properly to ensure that it is interpreted as data and not executable code.

**HttpOnly and Secure Flags:** Set the HttpOnly and Secure flags for cookies to prevent client-side script access to cookies.

**Session Management:** Implement secure session management practices to prevent session hijacking.

# 

# RCE

RCE stands for Remote Code Execution, which is a severe security vulnerability that allows attackers to execute arbitrary code on a remote system or web server. It occurs when an application or system does not properly validate and sanitize input data, allowing an attacker to inject malicious code that gets executed with the same privileges as the targeted application or service.

RCE can have disastrous consequences, as attackers can gain full control over the compromised system, leading to various attacks, such as data theft, unauthorized access, and further network exploitation.

## Common vectors for RCE include:

**Web Applications:** RCE vulnerabilities can exist in web applications, especially those written in languages like PHP, Python, Ruby, or Java. These vulnerabilities may arise from improper input validation, insecure deserialization, or command injection flaws.

**Remote Services:** Insecurely configured remote services, such as web servers (e.g., Apache, Nginx), databases (e.g., MySQL, PostgreSQL), and application servers (e.g., Tomcat, JBoss), can be exploited to achieve RCE.

**Exploiting Buffer Overflows:** In certain cases, RCE can be achieved by exploiting buffer overflow vulnerabilities in native applications that allow execution of arbitrary code.

## Impact of RCE:

**Unrestricted Access**: Attackers can gain complete control over the compromised system, enabling them to run arbitrary commands, modify files, and install malware.

**Data Exfiltration:** Sensitive data stored on the compromised system can be stolen, leading to potential data breaches.

**Lateral Movement:** Once attackers gain access to one system, RCE can enable them to pivot and move laterally across the network, compromising more systems.

**Service Disruption:** Attackers can execute destructive commands, leading to service disruption or system downtime.

## Preventing RCE:

**Input Validation and Sanitization:** Validate and sanitize all user inputs and data received from external sources to prevent malicious code injection.

**Secure Coding Practices:** Follow secure coding practices to avoid common vulnerabilities like command injection, deserialization flaws, and buffer overflows.

**Least Privilege Principle:** Ensure that applications and services have the least necessary privileges to perform their intended tasks.

**Regular Patching and Updates:** Keep all software, applications, and systems up to date with the latest security patches and updates to mitigate known vulnerabilities.

**Network Segmentation:** Segment networks to limit the spread of attacks and minimize the impact of potential RCE incidents.