Prevention Guide

Preventing SQL Injection Vulnerabilities: A developer's guide

Practical steps to keep your web applications and APIs safe and secure

Table of Contents

Defending Against SQL Injection Attacks

This guide explores SQL injection attacks, where hackers manipulate the application's SQL queries to gain unauthorized access to confidential information, tamper with the database, or disrupt the application's functionality. It covers different types of SQL injection attacks and provides practical insights on detecting and preventing them in modern web applications.

Understanding SQL Injection Vulnerabilities and Attacks

SQL injection is a code injection technique that allows hackers to send malicious SQL queries to a web application's backend database. This vulnerability occurs when the web page accepts user input as a SQL statement that the database executes. By exploiting this vulnerability, attackers can gain unauthorized access to sensitive data, including server configurations, user lists, and confidential company information that the application is not intended to reveal.

SQL injections are the most common type of web application attacks and are divided into various categories based on the techniques used to extract data. Types of SQL injections include:

1.

In-band SQL Injection Attacks

This type of SQL injection occurs when the hacker uses the same communication channel to send malicious SQL database queries and collect results. In-band SQL is considered the simplest and most prevalent form of SQL attack and falls into two main categories:

- **Error-based in-band SQL injection**
  In this type of attack, the hacker obtains information about the configuration of the database from error messages generated by the server. In addition, attackers can enumerate the entire database engine using malicious input that exposes information about its version and structure.

- **Union-based SQL injection**
  The attacker relies on the UNION operator to combine a genuine SQL query with the supplied malicious SELECT statements of the same structure to get a single HTTP response. The information to be accessed is part of this HTTP response. Therefore, the hacker can extend the results of an original query to extract further information about the database.

2.

Out-of-band SQL Injection Attacks

In this SQL injection attack, the malicious user cannot gather information from the same channel used to launch the attack. Instead, they rely on available functions to exfiltrate data through outbound channels. Such functions include connection establishment functions and file operations. The success of this attack depends on whether certain database server features are enabled, such as its ability to initiate an outbound DNS/HTTP request for data transfer, making the attack less prevalent.

3.

Inferential/blind SQL Injection Attacks

In this SQL injection attack, the application does not transfer data from the database to the attacker. As a result, hackers have to learn about the server's structure by observing its behavior and response. These attacks are much slower since successful attacks require continuous observation of HTTP response patterns. Blind SQL injection attacks are typically categorized into:

- **Boolean/content-based attacks**
  The attacker crafts malicious SQL statements that ask the database TRUE or FALSE questions, then validating whether these queries modify the information within the server's HTTP response. The attacker then makes inferences about the server's structure and configuration by determining if each message generates a TRUE or FALSE statement.

- **Time-based SQL injection attacks**
  The attacker crafts a query that forces the database server to wait for a set period before sending the response. This is followed by sending a malicious SQL payload and observing whether the server responds instantly or after a delay. Attackers use the response to infer whether a statement used is TRUE or FALSE, thus enumerating the database without relying on data in its response.

Severity Levels of SQL Injection Vulnerabilities

The impact and severity of an SQL injection attack depends on the security countermeasures in place and the attacker's skill. The Online Web Application Security Project (OWASP) recognizes injection attacks as one of the most common and far-reaching attacks, consistently ranking them as one of the top 10 application security risks. The attack is mainly found in application frameworks with legacy functional interfaces and is a common vulnerability of ASP and PHP web applications.

By injecting malicious commands into the data plane input, attackers influence how the application executes SQL commands while obtaining access to information stored and processed by the application. As a result, such attacks have severe effects, including:

1.

Identity Spoofing

Attackers use SQL injections to collect credentials of an application's users in the database. They can then impersonate the database user and carry out privileges and tasks according to the victim.

2.

Repudiation Issues

SQL injection vulnerabilities allow attackers access to data stored in a database server. As a result, they can alter the data in the database resulting in repudiation errors such as mismatched balance and null transactions, affecting the application's data credibility.

3.

Unauthorized Data Access

SQL injection vulnerability lets adversaries select and output database table names and column names, thereby allowing them to access sensitive intellectual data, personal data, or the database structure, granting them complete control of the application.

4.

Chained Attacks

In applications where the OS relies on the database server, attackers can use SQL injection attacks as the initial attack vector to orchestrate further attacks within the application.

Detecting SQL Injection Vulnerabilities

Veracode Dynamic Analysis is a dynamic application security testing solution that helps reduce the risk of being hacked through SQL injection vulnerabilities. The software establishes systematic tests across each phase of the software development lifecycle to uncover and mitigate SQL injection attack surfaces. **Veracode Dynamic Analysis** is built to find injection vulnerabilities in data-driven applications to eliminate blind spots before attackers can leverage them to feed malicious SQL commands to the backend database.

Veracode's solution relies on test cases developed using SQL payloads built to generate specific error messages. In addition, the tool includes an extensive list of queries and variables that enable security and QA teams to identify vulnerabilities of many SQL injection attack vectors.

The solution also blends Blind SQL injection with machine learning techniques and application security mechanisms (DAST) to emulate different attack patterns used in exploiting the vulnerability. This helps to uncover various approaches used by hackers to exploit the vulnerability and enables the system to proactively identify and defend against potential threats.

Veracode Dynamic Analysis natively integrates with all modern web development stacks, enabling teams to instantly scan web applications and APIs for SQL injection vulnerabilities. The quick security assessment compares the website against Veracode's comprehensive vulnerability database and OWASP's top 10 benchmarks to ensure the website is free of security gaps that are commonly used for SQL injection.

Preventing SQL Injection Vulnerabilities

To prevent SQL injection attacks, it is important to implement the following measures.

Enforce Server and Client-side Input Validation

Enforce Server and Client-side Input Validation: Implement robust validation techniques to ensure that users only submit allowed input types. Define policies that specify the acceptable length, style, input format, and other characteristics for valid input. Additionally, use whitelists to restrict input to specific types that pass validation checks, providing maximum control over the accepted input strings.

For structured data, developers should employ regular expressions to enforce strong input validation. This helps ensure that the input adheres to a predefined pattern or format. When possible, use pre-defined value sets such as drop-down lists to ensure that the input data matches the defined characteristics precisely. This reduces the risk of accepting unexpected or malicious input.

It is crucial to adopt a defensive mindset and treat all user-supplied input as potentially malicious. Validate and sanitize all data received from external sources to prevent unauthorized parties from executing malicious SQL statements.

Utilize Parameterized Queries

A parameterized query is a pre-built SQL statement that prompts the user to supply parameters (values/variables) before the server executes it. This approach allows developers to define all acceptable SQL code, making it easier for the database server to distinguish between user input data and executable code. By automatically quoting the parameter's value, a parameterized query ensures that user-supplied input does not subvert the application's logic, effectively preventing SQL injection attacks. Even if a hacker attempts to enter malicious SQL commands, the intent of the query remains unchanged, providing protection against the execution of unwanted statements.

Implement Stored Procedures

Stored procedures enable developers to define and store prepared SQL statements within the database server, making them easily accessible from the application. By grouping several SQL statements into a logical unit and defining the execution plan, these queries can be reused effortlessly. This approach ensures that the application only executes trusted SQL queries, effectively preventing SQL injection attempts. Furthermore, most development frameworks support parameterization of stored SQL statements, providing an additional layer of defense against SQL injection attacks.

Enforce the Law of Least Privilege

Developers should minimize the use of administrative privilege accounts when connecting the API to the database server, unless it is absolutely necessary. This precaution is crucial as attackers could exploit the database to gain access to the root system, enabling escalated attacks on the operating system and backend server. Furthermore, it is recommended to assign separate database access credentials for each application/entity, with minimal access rights to the hosted data tables. By implementing Role-Based Access Controls (RBAC), teams can assess the access requirements of each user and establish appropriate modes to grant only the necessary permissions.

Enable Object Relational Mapping (ORM)

ORM frameworks provide a seamless translation of SQL query results into code, eliminating the need for SQL statements in application code. Additionally, ORM mappers utilize parameterized statements to restrict the execution of user-supplied input. However, it's important to note that while ORM models prevent dynamic SQL queries, web development frameworks still allow fragmented SQL statements for complex operations, meaning the site is not entirely immune to attacks.

Escape User Input

As a final measure, developers should escape all user-supplied input strings and special characters before incorporating them into a query. By employing the appropriate escaping scheme for user input, the server can differentiate between input and code, effectively preventing the execution of unintended SQL commands.

Use a Web Application Firewall

SQL injection attacks can take on different patterns, and there are attack vectors that may not be covered by prevention methods like parameterization and user input validation. To provide comprehensive protection against various web security threats, including SQL injections, the Web Application Firewall (WAF) proves to be invaluable. By identifying and mitigating adversaries and malicious code before they reach the vulnerable web server, the WAF helps prevent attacks. Implementing a robust WAF allows developers, security teams, and quality assurance teams to enhance the application's defenses against SQL injection attacks without requiring significant changes to the application itself.

Best Practices to Prevent SQL Injection Attacks

To prevent SQL injection attacks in web frameworks, it is crucial to validate user input and avoid dynamic queries. In the upcoming section, we will explore the implementation of SQL injection prevention measures in popular development frameworks.

Django

Django simplifies SQL injection prevention by providing querysets that automatically construct parameterized queries. These querysets ensure that query parameters are defined separately from the query code. In more complex scenarios, Django allows developers to execute custom SQL and raw queries. To mitigate risks, it is recommended to utilize Django's authentication library for executing arbitrary queries using the extra() and RawSQL() constructs. Additionally, escaping input to these queries restricts the SQL input that users can provide. Django's Object Relational Model (ORM) further enhances security by mapping database tables with matching fields and implementing query parameterization, effectively eliminating the SQL injection attack surface.

Laravel

Laravel ships with the Fluent Query Builder and the Eloquent ORM, which facilitate the creation of prepared statements. PHP developers working with Laravel can utilize these features to prevent special characters from escaping through application forms. By leveraging these integrations, any malicious SQL queries passed through these forms are escaped, resulting in the database saving the invalid query as plain text.

Strengthen Your Web Applications and APIs Against Attacks

Veracode Dynamic Analysis is a comprehensive solution for dynamic application security testing. It enables continuous scanning and testing to uncover vulnerabilities in APIs, Javascript, and web applications. The solution provides actionable reports with low false positives and negatives, giving security teams confidence in identifying and addressing security gaps. The reports also include remediation advice, saving time in finding solutions to fix vulnerabilities.

Veracode Dynamic Analysis can be used alongside the Veracode's Software Security Platform, which integrates various tools and systems into your software development life cycle. This integration helps identify security risks across your entire tech stack. By combining Dynamic Analysis with Static Analysis and Software Composition Analysis, you can prevent the introduction of new flaws and reduce risk over time in your modern software development processes.

Try Veracode Dynamic Analysis for free with a 14-day trial and see how it can help strengthen your software against SQL injection attacks.

Start Your Free, 14-day Trial

VERACODE

Veracode is intelligent software security. The Veracode Software Security Platform continuously finds flaws and vulnerabilities at every stage of the modern software development lifecycle. Prompted by powerful AI trained by trillions of lines of code, Veracode customers fix flaws faster with high accuracy. Trusted by security teams, developers, and business leaders from thousands of the world's leading organizations, Veracode is the pioneer, continuing to redefine what intelligent software security means.

Learn more at **www.veracode.com**, on the **Veracode blog** and on **Twitter.**