CWE-89: SQL Injection

This guide will describe the SQL injection attack in more detail.

Contents:

- SQL Injection (CWE-89)
- SQL injection (CWE-89) coverage by Kiuwan

SQL Injection (CWE-89)

> **CWE-89** describes **SQL injection** as follows:
>
> "The software constructs all or part of an **SQL command** using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component."

Any SQL injection attack consists of insertion (or "injection") of malicious code within the SQL command executed by the app.

Effects of such malicious code injections can be unpredictable, depending on the attacker's intelligence and SQL-interpreter's characteristics, but the most common are:

- Read/modify sensitive data
- Execute administrative operations
- Execute commands on the underlying OS

> The most basic SQL injection attack is based on exploiting a dynamically constructed SQL query based on input data.

Let's suppose an app that displays the user's data based on the user's name as typed from the application user in a web form.

Dynamically constructed SQL in-app code might look something like:

```
"select * from users where name = '" + userName + "'";
```

userName is user-supplied data that is directly inserted into the query and it will be sent to SQL-engine to be executed.

Let's imagine the result when the attacker supplies the following text:

```
' or '1'='1
```

In this case, SQL-engine will return all users' data because 1=1 will always be TRUE.

This attack would the attacker be able to access users' data (involving a privacy breach), but consequences can be more serious combining query chaining with administrative commands such as

```
 Smith';drop table users; truncate audit_log;--
```

In this case, the attacker would be able to delete the *users* table or truncate system tables. Everything depends on the concrete case, but "the door is open" and, as you can imagine, imagination is the limit. How does the attacker know the app database tables? Depending on the error messages the application produces when a SQL injection attack happens, a smart attacker might be "inferring" database structure information from the error page. Discovering useful information is a matter of patience.

You could be thinking of a common app error management approach consisting of providing a generic error page, not displaying any exploitable information about the app's internals.

Even in this case, the app is still vulnerable to SQL injection. Attackers can use a technique known as **Blind SQL injection**.

This hacking technique is based on asking the database questions and determines the answer based on the application's response. This attack is often used when the web application is configured to show generic error messages but has not mitigated the code that is vulnerable to SQL injection.

Some variants of SQL injection apply to specific frameworks or conditions:

- CWE-564: SQL injection: Hibernate
- CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key

Find further information on SQL injection at https://www.kiuwan.com/blog/SQL injection-avoid/

SQL injection (CWE-89) coverage by Kiuwan

In Kiuwan, you can search for rules covering SQL injection (CWE-89) filtering by

- Vulnerability Type = **Injection**, and/or
- CWE tag = **CWE:89**

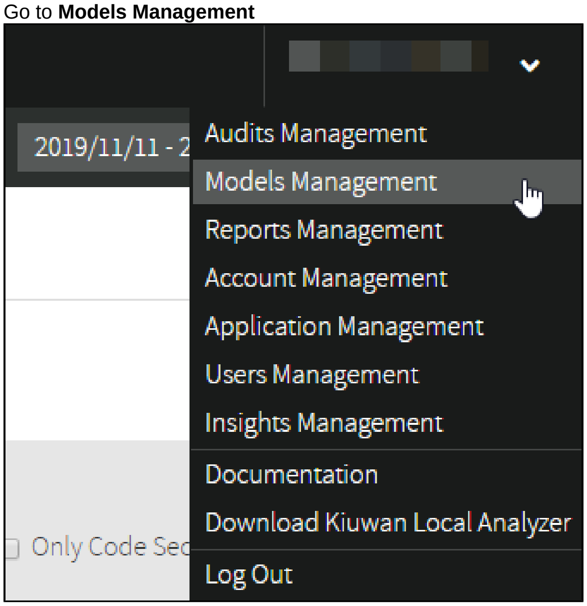
Kiuwan incorporates the following rules for SQL injection (CWE-89) for the following languages:

Language	Rule code
Abap	OPT.ABAP.SEC.SqlInjection
C#	OPT.CSHARP.SqlInjection
Cobol	OPT.COBOL.SEC.SqlInjection
Java	OPT.HIBERNATE.BindParametersInQueries
	OPT.JAVA.ANDROID.ContentProviderUriInjection
	OPT.JAVA.SEC_JAVA.IBatisSqlInjectionRule
	OPT.JAVA.SEC_JAVA.SqlInjectionRule
Javascript	OPT.JAVASCRIPT.SqlInjection
Objective-C	OPT.OBJECTIVEC.AvoidSqlInjection
Oracle Forms	OPT.ORACLEFORMS.SqlInjection
PHP	OPT.PHP.SqlInjection
Python	OPT.PYTHON.SECURITY.SqlInjection
RPG IV	OPT.RPG4.SEC.SqlInjection
Swift	OPT.SWIFT.SECURITY.SqlInjection
Transact-SQL	OPT.TRANSACTSQL.AvoidDynamicSql

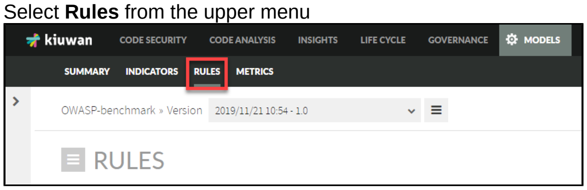
To read more detailed information about the rule on functionality, coverage, parameterization, remediation, example codes, etc., do the following:

1. Log into your Kiuwan Account

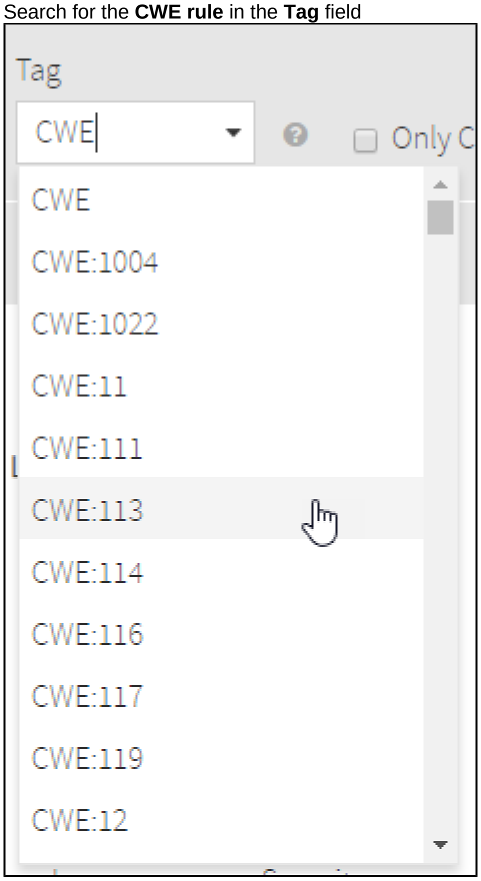
2. Go to **Models Management**



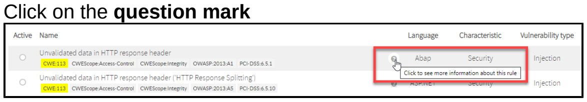
3. Select **Rules** from the upper menu



4. Search for the **CWE rule** in the **Tag** field



5. Click on the **question mark**



6. A new window will open with more detailed information about the rule