




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 **Java**


 JavaScript


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6

 XML



## Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects

Bug

"wait" should not be called when multiple locks are held

Bug

"PreparedStatement" and "ResultSet" methods should be called with valid indices

Bug

Files opened in append mode should not be used with ObjectOutputStream

Bug

"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held

Bug

Printf-style format strings should not lead to unexpected behavior at runtime

Bug

Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances

Bug

Methods should not call same-class methods with incompatible "@Transactional" values

Bug

Recursion should not be infinite

Bug

Loops should not be infinite

Bug

Double-checked locking should not be used

Database queries should not be vulnerable to injection attacks

Analyze your code

VulnerabilityBlocker🔍 injection cwe spring owasp sans-top25 cert hibernate sql

User-provided data, such as URL parameters, should always be considered untrusted and tainted. Constructing SQL queries directly from tainted data enables attackers to inject specially crafted values that change the initial meaning of the query itself. Successful database query injection attacks can read, modify, or delete sensitive information from the database and sometimes even shut it down or execute arbitrary operating system commands.

Typically, the solution is to use prepared statements and to bind variables to SQL query parameters with dedicated methods like `setString`, which ensures that user-provided data will be properly escaped. Another solution is to validate every parameter used to build the query. This can be achieved by transforming string values to primitive types or by validating them against a white list of accepted values.

This rule supports: JDBC, Java EE Entity Manager, Spring Framework, Hibernate, JDO, Android Database, Apache Torque, Apache Turbine, MyBastis, Rapidoid.

Noncompliant Code Example

```
public boolean authenticate(javax.servlet.http.HttpServletRe
String user = request.getParameter("user");
String pass = request.getParameter("pass");

String query = "SELECT * FROM users WHERE user = '" + user

// If the special value "foo' OR 1=1 --" is passed as eith
// Indeed, if it is passed as a user, the query becomes:
// SELECT * FROM users WHERE user = 'foo' OR 1=1 --' AND p
// As '--' is the comment till end of line syntax in SQL,
// SELECT * FROM users WHERE user = 'foo' OR 1=1
// which is equivalent to:
// SELECT * FROM users WHERE 1=1
// which is equivalent to:
// SELECT * FROM users

java.sql.Statement statement = connection.createStatement(
java.sql.ResultSet resultSet = statement.executeQuery(quer
return resultSet.next();
}
```

Compliant Solution






```
public boolean authenticate(javax.servlet.http.HttpServletRe
String user = request.getParameter("user");
String pass = request.getParameter("pass");

String query = "SELECT * FROM users WHERE user = ? AND pas

java.sql.PreparedStatement statement = connection.prepareS
```

https://rules.sonarsource.com/java/RSPEC-3649

1/2

 Bug
Resources should be closed
 Bug
Hard-coded credentials are security-sensitive
 Security Hotspot
Methods returns should not be invariant
 Code Smell
"ThreadGroup" should not be used
 Code Smell

```
statement.setString(1, user); // Will be properly escaped
statement.setString(2, pass);
java.sql.ResultSet resultSet = statement.executeQuery();
return resultSet.next();
}
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-89](#) - Improper Neutralization of Special Elements used in an SQL Command
- [MITRE, CWE-564](#) - SQL Injection: Hibernate
- [MITRE, CWE-943](#) - Improper Neutralization of Special Elements in Data Query Logic
- OWASP SQL Injection Prevention [Cheat Sheet](#)
- [CERT, IDS00-J](#). - Prevent SQL injection
- [SANS Top 25](#) - Insecure Interaction Between Components

Available In:

  Developer Edition