




 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





JavaScript static code analysis


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


All rules285

 Vulnerability29

 Bug62


 Security Hotspot43

 Code Smell151


 Quick Fix41

Tags ▾


Search by name... 🔍

 Code Smell


Unnecessary imports should be removed




Return of boolean expressions should not be wrapped into an "if-then-else" statement




Boolean literals should not be used in comparisons




Extra semicolons should be removed




Class names should comply with a naming convention




Track uses of "TODO" tags




Web SQL databases should not be used




Variables should be defined before being used




Variables declared with "var" should be declared before they are used





Track lack of copyright and license headers



Reading the Standard Input is security-sensitive



Formatting SQL queries is security-sensitive

 Security Hotspot Major ⓘ

cwe owasp sans-top25 bad-practice sql

Formatted SQL queries can be difficult to maintain, debug and can increase the risk of SQL injection when concatenating untrusted values into the query. However, this rule doesn't detect SQL injections (unlike rule {rule:javascript:S3649}), the goal is only to highlight complex/formatted queries.

Ask Yourself Whether

- Some parts of the query come from untrusted values (like user inputs).
- The query is repeated/duplicated in other parts of the code.
- The application must support different types of relational databases.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Use parameterized queries, prepared statements, or stored procedures and bind variables to SQL query parameters.
- Consider using ORM frameworks if there is a need to have an abstract layer to access data.

Sensitive Code Example

```
// === MySQL ===
const mysql = require('mysql');
const mycon = mysql.createConnection({ host: host, user: use
mycon.connect(function(err) {
  mycon.query('SELECT * FROM users WHERE id = ' + userInput,
});

// === PostgreSQL ===
const pg = require('pg');
const pgcon = new pg.Client({ host: host, user: user, passwo
pgcon.connect();
pgcon.query('SELECT * FROM users WHERE id = ' + userInput, (
```





Compliant Solution

```
// === MySQL ===
const mysql = require('mysql');
const mycon = mysql.createConnection({ host: host, user: use
mycon.connect(function(err) {
  mycon.query('SELECT name FROM users WHERE id = ?', [userin
});

// === PostgreSQL ===
const pg = require('pg');
const pgcon = new pg.Client({ host: host, user: user, passwo
```

https://rules.sonarsource.com/javascript/RSPEC-2077

1/2

Using command line arguments is security-sensitive
 Security Hotspot
Using Sockets is security-sensitive
 Security Hotspot
Executing XPath expressions is security-sensitive
 Security Hotspot
Encrypting data is security-sensitive
 Security Hotspot
Using regular expressions is security-sensitive

```
pgcon.connect();
pgcon.query('SELECT name FROM users WHERE id = $1', [usingip
```

Exceptions

This rule's current implementation does not follow variables. It will only detect SQL queries which are formatted directly in the function call.

```
const sql = 'SELECT * FROM users WHERE id = ' + userInput;
mycon.query(sql, (err, res) => {}); // Sensitive but no issu
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-89](#) - Improper Neutralization of Special Elements used in an SQL Command
- [MITRE, CWE-564](#) - SQL Injection: Hibernate
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-943](#) - Improper Neutralization of Special Elements in Data Query Logic
- [SANS Top 25](#) - Insecure Interaction Between Components
- Derived from FindSecBugs rules [Potential SQL/JPQL Injection \(JPA\)](#), [Potential SQL/JDOQL Injection \(JDO\)](#), [Potential SQL/HQL Injection \(Hibernate\)](#)

Available In:  