

sonar


RULES


Products


▼


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...

Regular expressions using Unicode character classes or property escapes should enable the unicode flag

Bug

The base should be provided to "parseInt"

Bug

Function declarations should not be made within blocks

Bug

Writing cookies is security-sensitive

Security Hotspot

"continue" should not be used

Code Smell

Trailing commas should be used

Code Smell

"import" should be used to include external code

Code Smell

Braces and parentheses should be used consistently with arrow functions

Code Smell

Destructuring syntax should be used for assignments

Code Smell

Template strings should be used instead of concatenation

Code Smell

Shorthand object properties should be grouped at the beginning or end of an object declaration

Code Smell

Object literal shorthand syntax should be used

### Two branches in a conditional structure should not have exactly the same implementation

Analyze your code

Code Smell

Major

design suspicious

Having two cases in a switch statement or two branches in an if chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an if chain they should be combined, or for a switch, one should fall through to the other.

#### Noncompliant Code Example

```
switch (i) {
  case 1:
    doFirstThing();
    doSomething();
    break;
  case 2:
    doSomethingDifferent();
    break;
  case 3: // Noncompliant; duplicates case 1's implementation
    doFirstThing();
    doSomething();
    break;
  default:
    doTheRest();
}

if (a >= 0 && a < 10) {
  doFirstThing();
  doTheThing();
}
else if (a >= 10 && a < 20) {
  doTheOtherThing();
}
else if (a >= 20 && a < 50) {
  doFirstThing();
  doTheThing(); // Noncompliant; duplicates first condition
}
else {
  doTheRest();
}
```






#### Exceptions

Blocks in an if chain that contain a single line of code are ignored, as are blocks in a switch statement that contain a single line of code with or without a following break.

```
if (a == 1) {
  doSomething(); //no issue, usually this is done on purpose
} else if (a == 2) {
  doSomethingElse();
} else {
```

https://rules.sonarsource.com/javascript/RSPEC-1871

1/2

 Code Smell
<b>Strings and non-strings should not be added</b>  Code Smell
<b>Object literal syntax should be used</b>  Code Smell
<b>"undefined" should not be assigned</b>  Code Smell
<b>Trailing commas should not be used</b>  Code Smell

```
doSomething();
}
```

But this exception does not apply to `if` chains without `else`-s, or to `switch`-es without default clauses when all branches have the same single line of code. In case of `if` chains with `else`-s, or of `switch`-es with default clauses, rule {rule:javascript:S3923} raises a bug.

```
if (a == 1) {
  doSomething(); //Noncompliant, this might have been done
} else if (a == 2) {
  doSomething();
}
```

Available In:

**sonarlint**  | **sonarcloud**  | **sonarqube** 