sonar RULES

Products

Swift static code analysis

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

All rules 119 | Vulnerability 3 | Bug 14 | Security Hotspot 3 | Code Smell 99

Tags | Search by name...

Hard-coded credentials are security-sensitive
Security Hotspot

Methods and field names should not be the same or differ only by capitalization
Code Smell

Cipher algorithms should be robust
Vulnerability

Using weak hashing algorithms is security-sensitive
Security Hotspot

Cognitive Complexity of functions should not be too high
Code Smell

"try!" should not be used
Code Smell

String literals should not be duplicated
Code Smell

Functions and closures should not be empty
Code Smell

Collection elements should not be replaced unconditionally
Bug

Collection sizes comparisons should make sense
Bug

All branches in a conditional structure should not have exactly the same implementation
Bug

Infix operators that end with "=" should update their left operands
Bug

Precedence and associativity of standard operators should not be changed

Identical expressions should not be used on both sides of a binary operator

Analyze your code

Bug Major

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code, and should be simplified. In the case of bitwise operators and most binary mathematical operators, having the same value on both sides of an operator yields predictable results, and should be simplified.

This rule ignores `*`, `+`.

Noncompliant Code Example

```
if a == a { // always true
  doZ()
}
if  a != a  { // always false
  doY()
}
if a == b && a == b { // if the first one is true, the s
  doX()
}
if a == b || a == b { // if the first one is true, the s
  doW()
}

var j = 5 / 5 //always 1
var k = 5 - 5 //always 0
```

Exceptions

Left-shifting 1 onto 1 is common in the construction of bit masks, and is ignored.

```
var i = 1 << 1; // Compliant
var j = a << a; // Noncompliant
```

See

- {rule:swift:S1656} - Implements a check on =.

Available In:
sonarlint sonarcloud sonarqube Developer Edition

Bug

Return values from functions without side effects should not be ignored

Bug

Related "if/else if" statements and "cases" in a "switch" should not have the same condition

Bug

Identical expressions should not be used on both sides of a binary operator

Bug

All code should be reachable

Bug

Loops with at most one iteration should be refactored

Bug

"IBInspectable" should be used correctly

Code Smell

Functions should not have identical implementations

Code Smell

Ternary operators should not be nested

Code Smell

Closure expressions should not be nested too deeply

Code Smell

Backticks should not be used around