sonar RULES

Products

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Kubernetes
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- **Swift**
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML

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

Floating point numbers should not be tested for equality

Analyze your code

Bug | Major

Floating point math is imprecise because of the challenges of storing such values in a binary representation. Even worse, floating point math is not associative; push a `Float` or a `Double` through a series of simple mathematical operations and the answer will be different based on the order of those operation because of the rounding that takes place at each step.

Even simple floating point assignments are not simple:

```
var f: Float = 0.1 // 0.1000000014901161193847656
var d: Double = 0.1 // 0.1000000000000000055511151
```

Therefore, the use of the equality (`==`) and inequality (`!=`) operators on `Float` or `Double` values is almost always an error.

This rule checks for the use of direct and indirect equality/inequailty tests on floats and doubles.

Noncompliant Code Example

```
var myNumber: Float = 0.3 + 0.6

if myNumber == 0.9 { // Noncompliant. Because of floating p
    // ...
}

if myNumber <= 0.9 && myNumber >= 0.9 { // Noncompliant inc
  // ...
}

if myNumber < 0.9 || myNumber > 0.9 { // Noncompliant indir
  // ...
}
```

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

Precedence and associativity of standard operators should not be changed

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