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

Precedence and associativity of standard operators should not be changed

Functions and variables should not be defined outside of classes

Analyze your code

Code Smell | Blocker | design

Defining and using global variables and global functions, when the convention dictates OOP can be confusing and difficult to use properly for multiple reasons:

- You run the risk of name clashes.
- Global functions must be stateless, or they can cause difficult-to-track bugs.
- Global variables can be updated from anywhere and may no longer hold the value you expect.
- It is difficult to properly test classes that use global functions.

Instead of being declared globally, such variables and functions should be moved into a class, potentially marked `static`, so they can be used without a class instance.

This rule checks that only object-oriented programming is used and that no functions or procedures are declared outside of a class.

Noncompliant Code Example

```
var name = "Bob"    // Noncompliant

func doSomething() {   // Noncompliant
  //...
}

class MyClass {
    //...
}
```

Compliant Solution

```
public class MyClass {
  public static var name = "Bob"

  public class func doSomething() {               // Comp
    //...
  }
  //...
}
```

Exceptions

The operator function is a function with a name that matches the operator to be overloaded. Because such functions can only be defined in a global scope, they are ignored by this rule.

```
public class Vector2D {
    var x = 0.0, y = 0.0
    // ...
}

func + (left: Vector2D, right: Vector2D) -> Vector2D {
    return Vector2D(x: left.x + right.x, y: left.y + rig
```

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

```
}
```

Available In:

sonarlint sonarcloud sonarqube Developer Edition