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

Operator functions should call existing functions

Analyze your code

Code Smell Minor convention api-design

Making an operator a convenience wrapper around an existing function or method provides additional flexibility to users in how the functionality is called and in what options are passed in.

This rule raises an issue when the function that defines the operation of a operator consists of something other than a single function call.

Noncompliant Code Example

```
infix operator >< { associativity right precedence 90 }
func >< (left: Double, right: Double) -> Double {  // No
  let leftD = (left % 1) * 100
  let rightD = (right % 1) * 100
  let leftW = (left - leftD) / 100
  let rightW = (right - rightD) / 100
  return (leftD + leftW) * (rightD + rightW)
}
```

Compliant Solution

```
infix operator >< { associativity right precedence 90 }
func >< (left: Double, right: Double) -> Double {
  return fubar(left, right)
}

func fubar(left: Double, right: Double) -> Double {
  let leftD = (left % 1) * 100
  let rightD = (right % 1) * 100
  let leftW = (left - leftD) / 100
  let rightW = (right - rightD) / 100
  return (leftD + leftW) * (rightD + rightW)
}
```

Exceptions

Operators that end with = are expected to update their left-hand operands, and are therefore ignored.

```
func **= (inout p1:Int, p2:Int) {
    p1 = p1 ** p2
}
```

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