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
Bug

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 the same `switch` statement or branches in the same `if` structure 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 they should be combined.

Noncompliant Code Example

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

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

Exceptions

`case` labels that declare variables cannot have multiple patterns. Therefore this situation is ignored.

```
switch a {
    case .STR_CASE(let x):
        print(x)
    case .INT_CASE(let x):
        print(x)
    default:
        print("default")
}
```

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 >= 0 && a < 10 {    //no issue, usually this is done or
  doTheThing()
} else if a >= 10 && a < 20 {
  doTheThing()
} else if a >= 20 && a < 50 {
  doFirstThing()
}
```

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:swift:S3923} raises a bug.

```
if a >= 0 && a < 10 {     //Noncompliant, this might have be
  doTheThing()
} else if a >= 10 && a < 20 {
  doTheThing()
}
```

Available In:

sonarlint sonarcloud sonarqube Developer Edition

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 symbol names