

-  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	
Methods and field names should not be the same or differ only by capitalization	
Cipher algorithms should be robust	
Using weak hashing algorithms is security-sensitive	
Cognitive Complexity of functions should not be too high	
"try!" should not be used	
String literals should not be duplicated	
Functions and closures should not be empty	
Collection elements should not be replaced unconditionally	
Collection sizes comparisons should make sense	
All branches in a conditional structure should not have exactly the same implementation	
Infix operators that end with "=" should update their left operands	
Precedence and associativity of standard operators should not be changed	

Implicitly unwrapped optionals should not be used

Analyze your code

-  Bug
-  Critical
- 
-  pitfall

The point of using an optional is to signal that the value may be `nil` and to provide graceful ways of dealing with it if it is `nil`. While implicitly unwrapped optionals still provide means of dealing with `nil` values, they also signal that the value won't be `nil`, and unwrap it automatically. In addition to sending a decidedly mixed signal, this could lead to runtime errors if the value ever is `nil`.

It is safest, and clearest to use either an optional or a plain type and avoid the boggy middle ground of implicitly unwrapped optionals.

Noncompliant Code Example

```
var greeting : String! // Noncompliant

println(greeting) // At this point the value is nil. Runtime error!
```

Compliant Solution

```
var greeting : String?

if let howdy = greeting {
    println(howdy)
}
```

Available In:

 |  |  Developer Edition

 Bug
<div>Return values from functions without side effects should not be ignored</div> <div> Bug</div>
<div>Related "if/else if" statements and "cases" in a "switch" should not have the same condition</div> <div> Bug</div>
<div>Identical expressions should not be used on both sides of a binary operator</div> <div> Bug</div>
<div>All code should be reachable</div> <div> Bug</div>
<div>Loops with at most one iteration should be refactored</div> <div> Bug</div>
<div>"IBInspectable" should be used correctly</div> <div> Code Smell</div>
<div>Functions should not have identical implementations</div> <div> Code Smell</div>
<div>Ternary operators should not be nested</div> <div> Code Smell</div>
<div>Closure expressions should not be nested too deeply</div> <div> Code Smell</div>
<div>Backticks should not be used around</div>