

































-  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



Objective C static code analysis


Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your OBJECTIVE C code

All rules 315

 Vulnerability 10

 Bug 75

 Security Hotspot 18

 Code Smell 212

 Quick Fix 13

Tags

Search by name...



"memset" should not be used to delete sensitive data

 Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows

 Vulnerability

Function-like macros should not be invoked without all of their arguments

 Bug

The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist

 Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked

 Bug

"pthread_mutex_t" should be properly initialized and destroyed

 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

 Bug

Functions with "noreturn" attribute should not return

 Bug

"memcpy" should only be called with pointers to trivially copyable types with no padding

 Bug

Stack allocated memory and non-owned memory should not be freed

 Bug

Closed resources should not be accessed



 Bug

Dynamically allocated memory should be released

 Bug

Conditionals should start on new lines

Analyze your code

 Code Smell  Critical  suspicious

Code is clearest when each statement has its own line. Nonetheless, it is a common pattern to combine on the same line an `if` and its resulting *then* statement. However, when an `if` is placed on the same line as the closing `}` from a preceding *then*, *else* or *else if* part, it is either an error - `else` is missing - or the invitation to a future error as maintainers fail to understand that the two statements are unconnected.

Noncompliant Code Example

```
if (condition1) {
    // ...
} if (condition2) { // Noncompliant
    //...
```

Compliant Solution

```
if (condition1) {
    // ...
} else if (condition2) {
    //...
```

Or

```
if (condition1) {
    // ...
}

if (condition2) {
    //...
```

Available In:

sonarcloud  | sonarqube  Developer Edition

<div>Freed memory should not be used</div> <div> Bug</div>
<div>Memory locations should not be released more than once</div> <div> Bug</div>
<div>Memory access should be explicitly bounded to prevent buffer overflows</div> <div> Bug</div>
<div>Printf-style format strings should not lead to unexpected behavior at runtime</div> <div> Bug</div>
<div>Recursion should not be infinite</div> <div> Bug</div>
<div>Resources should be closed</div> <div> Bug</div>
<div>Hard-coded credentials are security-sensitive</div> <div> Security Hotspot</div>
<div>"goto" should jump to labels declared later in the same function</div> <div> Code Smell</div>
<div>Only standard forms of the "defined" directive should be used</div> <div> Code Smell</div>
<div>Switch labels should not be nested inside non-switch blocks</div> <div> Code Smell</div>