


-  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

"enum" values should not be used as operands to built-in operators other than [], =, ==, !=, unary &, and the relational operators <, <=, >, >=

Analyze your code

 Code Smell  Major  based-on-misra suspicious

Enumerations are used to represent symbolic values, or sometimes bit fields. They are not supposed to be used in arithmetic contexts.

Additionally, even though comparing them with integer numbers can make sense (for instance, to test if an enum lies with a certain range), comparing them with floating point numbers does not (and is deprecated since C++20).

There are other restrictions related to the use of enums, see for instance {rule:cpp:S2753}.

Noncompliant Code Example

```
enum { COLOUR_0, COLOUR_1, COLOUR_2, COLOUR_COUNT } colour;
if ( COLOUR_0 == colour ) { ... }
if ( ( COLOUR_0 + COLOUR_1 ) == colour ) { ... } // Noncompliant
if ( colour < COLOUR_COUNT ) { ... }
if ( colour > 3.14 ) { ... } // Noncompliant, comparison with
```

See

- MISRA C++:2008, 4-5-2 - Expressions with type enum shall not be used as operands to builtin operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=

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>