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

"memcmp" 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

Freed memory should not be used

Bug

Memory locations should not be released more than once

Bug

Memory access should be explicitly bounded to prevent buffer overflows

Bug

Printf-style format strings should not lead to unexpected behavior at runtime

Bug

Recursion should not be infinite

Bug

Resources should be closed

Bug

Hard-coded credentials are security-sensitive

Security Hotspot

"goto" should jump to labels declared later in the same function

Code Smell

Only standard forms of the "defined" directive should be used

Code Smell

Switch labels should not be nested inside non-switch blocks

Code Smell

"bool" expressions should not be used as operands to built-in operators other than =, &&, ||, !, ==, !=, unary &, and the conditional operator

Analyze your code

Code Smell Major based-on-misra suspicious

The use of `bool` operands with other operators is unlikely to be meaningful (or intended). Best case it will be confusing to maintainers, worst case it will not have the intended effect. Either way, it is highly recommended to stick to boolean operators when dealing with `bool` operands.

This rule allows the detection of such uses, which often occur because the logical operators (`&&`, `||` and `!`) can be easily confused with the bitwise operators (`&`, `|` and `~`).

Noncompliant Code Example

```
bool b1 = true;
bool b2 = false;
int8_t s8a;
if ( b1 & b2 ) // Noncompliant
if ( ~b1 ) // Noncompliant
if ( b1 < b2 ) // Noncompliant
if ( b1 ^ b2 ) // Noncompliant
```

Compliant Solution

```
if ( b1 && b2 )
if ( !b1 )
if ( b1 == false )
if ( b1 == b2 )
if ( b1 != b2 )
s8a = b1 ? 3 : 7;
```

Exceptions

Operators `|=` and `&=` are ignored when used with `bool` operands. Operator `++` is also ignored with a `bool` operand because it is covered by rule {rule:cpp:S2668}.

```
void test(bool b1, bool b2, int i1) {
  b1 |= b2; // ignored
  b1++; // ignored here, handled by S2668
  b1 &= b2; // ignored
  b1 &= i1; // Noncompliant; right operand is not a bool
}
```

See

- MISRA C++:2008, 4-5-1 - Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.

Available In:

sonarcloud sonarqube Developer Edition