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

"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

"volatile" should not be used to qualify objects for which the meaning is not defined

Analyze your code

🐛 Bug   🔺 Critical ⃠

`volatile` can be used to qualify many objects in C and C++, but only a few of the possible places have a well-defined meaning (global variables and local variables for instance). There is no well-defined meaning to the use of volatile to qualify a function return type or a function parameter. Furthermore, for structured bindings, the volatile qualifier appertains to the decomposed object which cannot be referred to. Since C++20, these uses are deprecated, but even before you should not use volatile in those places.

This rule raises an issue for a volatile qualified function return type, function parameter, and structured binding (available in C++ since C++17).

Noncompliant Code Example

```
int volatile f(int volatile i); // Noncompliant, both for the

void g() {
    auto volatile [a, b] = getPair(); // Noncompliant
}
```

Available In:

sonarcloud | sonarqube Developer Edition

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