

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- 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

Variables should not be shadowed

Analyze your code

Code Smell Major based-on-misra cert suspicious pitfall

Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another.

Noncompliant Code Example

```
class Foo
{
public:
    void doSomething();

private:
    int myField;
};

void Foo::doSomething()
{
    int myField = 0; // Noncompliant
    // ...
}
```

```
void f(int x, bool b) {
    int y = 4;
    if (b) {
        int x = 7; // Noncompliant
        int y = 9; // Noncompliant
        // ...
    }
}
```

Compliant Solution

```
class Foo
{
public:
    void doSomething();

private:
    int myField;
};

void Foo::doSomething()
{
    int myInternalField = 0; // Compliant
    // ...
}
```

```
void f(int x, bool b) {
    int y = 4;
    if (b) {
        int z = 7; // Better yet: Use meaningful names
        int w = 9;
        // ...
    }
}
```

<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>

```
}
}
```

Exceptions

It is common in a constructor to have constructor arguments shadowing the fields that they will initialize. This pattern avoids the need to select new names for the constructor arguments, and will not be reported by this rule:

```
class Point{
public:
    Point(int x, int y) : x(x), y(y) {} // Compliant by exception
private:
    int x;
    int y;
};
```

See

- MISRA C:2004, 5.2 - Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier
- MISRA C++:2008, 2-10-2 - Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope
- MISRA C:2012, 5.3 - An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
- [CERT, DCL01-C](#). - Do not reuse variable names in subscopes
- [CERT, DCL51-J](#). - Do not shadow or obscure identifiers in subscopes

Available In:

sonarcloud



sonarqube

 Developer Edition