

- 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

## GNU extensions should not be used

Analyze your code

Code Smell Minor lock-in obsolete gnu

Proprietary compiler extensions can be handy, but they commit you to always using that compiler. This rule raises an issue when GNU extensions are used, such as:

- Ternary operator with omitted second operand
- Case ranges in switch statements
- Expression statements, i.e. code blocks producing value
- Index range in array initializers
- A array initializer without =
- A structure member initializer with a colon
- Decimal floating points numbers `_Decimal132`, `_Decimal164`, and `_Decimal128`
- Structures and union without named data members

### Noncompliant Code Example

```
struct S {
    int f;
};

struct S s[] = {
    [0] { // Noncompliant
        f : 0 // Noncompliant
    }
    [1 ... 3] = { // CHECK :8 :11 S3715:use of GNU array range
        .f = 2
    }
};

int fun(int p) {
    switch (p) {
        case 0 ... 1: // Noncompliant
            do_the_thing();
            break;
        case 2:
            //...
    }

    p = ({ // Noncompliant
        int a = 10, b = 20;
        (a * b) + 10;
    });

    return p ?: 0; // Noncompliant
}

_Decimal132 d32; // Noncompliant

struct Empty {}; // Noncomplaint in C
```

### Compliant Solution

```
struct S {
    int f;
};

struct S s[] = {
    [0] = {
```

<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>

```
        .f = 0
    },
    [1] = {
        .f = 2
    }
    [2] = {
        .f = 2
    },
    [3] = {
        .f = 2
    }
};

int fun(int p) {
    switch (p) {
        case 0:
        case 1:
            do_the_thing();
            break;
        case 2:
            //...
    }

    int a = 10, b = 20;
    p = (a * b) + 10;

    return p ? p: 0;
}
```

Available In:

sonarcloud



sonarqube

 Developer Edition