



Apex

ABAP

С

C++

CloudFormation

COBOL

C#

CSS

Flex

=GO

HTML 5

Go

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 315 rules

6 Vulnerability 10

R Bug (75)

o Security Hotspot

⊗ Code (212)

O Quick 13 Fix

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 nonowned memory should not be freed

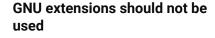
📆 Bug

Closed resources should not be accessed

📆 Bug

Dynamically allocated memory should be released

📆 Bug



Analyze your code

☼ Code Smell ♥ Minor ②

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 Decimal32, Decimal64, and
- 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
_Decimal32 d32; // Noncomplaint
struct Empty {}; // Noncomplaint in C
```

Compliant Solution

```
struct S {
  int f;
struct S s[] = {
  [0] = {
```

Freed memory should not be used

Memory locations should not be released more than once

📆 Bug

📆 Bug

Memory access should be explicitly bounded to prevent buffer overflows

📆 Bug

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

Rug Bug

Recursion should not be infinite

📆 Bug

Resources should be closed

📆 Bug

Hard-coded credentials are securitysensitive

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

```
},
  [1] = {
  [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

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved. **Privacy Policy**