



C static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C code

All rules **311**

Vulnerability **13**

Bug **74**

Security Hotspot **18**

Code Smell **206**

Quick Fix **14**

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

XML parsers should not be vulnerable to XXE attacks

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

"memcpy" should only be called with pointers to trivially copyable types with no padding

Analyze your code

Bug Blocker unpredictable

The function `memcpy` can only be used for objects of trivially copyable types. This includes scalar types, arrays, and trivially copyable classes.

A class type is trivially copyable if:

- One or more of the following methods is trivial and the rest are deleted: copy constructor, move constructor, copy assignment operator, and move assignment operator,
- It has a trivial, non-deleted destructor.

Additionally, if the type contains padding, some of its bits might be non-representative, and a strict comparison of raw memory contents might lead to the mistaken belief that two identical objects are actually different.

Noncompliant Code Example

```
class Shape { // Trivially copyable, but will contain padding
public:
    bool visible;
    int x;
    int y;
};

bool isSame(Shape *s1, Shape *s2)
{
    return memcpy(s1, s2, sizeof Shape) == 0; // Noncompliant
}
```

Compliant Solution

```
class Shape {
public:
    bool visible;
    int x;
    int y;
};

bool operator==(Shape const &s1, Shape const &s2) {
    return s1.visible == s2.visible && s1.x == s2.x && s1.y == s2.y;
}

bool isSame(Shape *s1, Shape *s2)
{
    return (*s1) == (*s2);
}
```

Available in:

sonarlint | sonarcloud | sonarqube Developer Edition

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

© 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](#)