

- 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



## 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

### User-defined types should not be passed as variadic arguments

Analyze your code

Bug Major suspicious

Variadic arguments allow a function to accept any number of arguments (in this rule, we are not talking about variadic templates, but about functions with ellipses). But these arguments have to respect some criteria to be handled properly.

This rules reports an issue if the type of the argument:

- is a non trivially copyable, movable or deletable type: there is no guarantee that it will work.
- is a class type that is trivially copyable, movable and deletable: in this case, we consider that the user intention was probably not to directly pass it as an argument but to call a method on it (`c_str()` for example)

#### Noncompliant Code Example

```
class A {
    char* toStr();
};
void v(...);

void f() {
    A a;
    v(a); // Noncompliant

    std::string myString = "foo";
    v(myString); // Noncompliant; string is not a POD type
}
```

#### Compliant Solution

```
class A {
    char* toStr();
}
void v(...);

void f() {
    A a;
    v(a.toStr()); // Compliant

    std::string myString = "foo";
    v(myString.c_str()); // Compliant
}
```

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**

SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)