

- 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

The prefix increment/decrement form should be used

Analyze your code

Code Smell Minor performance

Postfix increment and decrement typically involves making a copy of the object being incremented or decremented, whereas its prefix form does not. Therefore the prefix form is usually the more efficient form, and should be preferred.

This rule raises an issue if a postfix increment or decrement operator is used, but its return value is not read.

Noncompliant Code Example

```
void myFunc(int lim)
{
    int i;
    for (i = 0; i < lim; i++)
    {
        // do something
    }
}
```

Compliant Solution

```
void myFunc(int lim)
{
    int i;
    for (i = 0; i < lim; ++i)
    {
        // do something
    }
}
```

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