

- 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

Pointer and reference local variables should be "const" if the corresponding object is not modified

Analyze your code

Code Smell



Minor



Quick Fix



based-on-misra bad-practice

This rule leads to greater precision in the definition of local variables by making the developer intention about modifying the variable explicit. The `const` qualification shall be applied to the object pointed to, not to the pointer, since it is the object itself that is being protected.

Noncompliant Code Example

```
std::string& getString();
void myfunc()
{
    std::string& s = getString(); // Noncompliant
    if (s.size()) {
        std::cout << s;
    }
}
```

Compliant Solution

```
std::string& getString();
void myfunc () {
    const std::string& x = getString();
    if (s.size()) {
        std::cout << s;
    }
}
```

See

- MISRA C:2012, 8.13 - A pointer should point to a const-qualified type whenever possible

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