



ABAP

Apex

C

C++

CloudFormation

COBOL

C#

CSS

Flex

=GO

5

Go

Java

JavaScript

HTML

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

578 ΑII 6 Vulnerability 13 rules

R Bug (111)

• Security Hotspot

⊗ Code (436)

Quick 68 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

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

Assigning to an optional should directly target the optional

🖷 Bug

Result of the standard remove algorithms should not be ignored

📆 Bug

"std::scoped_lock" should be created with constructor arguments

📆 Bug

Objects should not be sliced

📆 Bug

Immediately dangling references should not be created

📆 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

Reserved identifiers and functions in the C standard library should not be defined or declared

Analyze your code





based-on-misra bad-practice cert

Defining or declaring identifiers with reserved names may lead to undefined behavior. Similarly, defining macros, variables or functions/methods with the same names as functions from the C standard library is likely to lead to unexpected results.

Additionally, such identifiers have the potential to thoroughly confuse people who are unfamiliar with the code base, possibly leading them to introduce additional errors. Therefore reserved words and the names of C standard library functions should not be used as identifiers.

This rule applies to:

- defined
- C standard library function names
- · identifiers that contain two consecutive underscores
- identifiers that begin with an underscore, followed by an uppercase letter
- · identifiers in the global namespace that start with an underscore

Noncompliant Code Example

```
#ifndef _MY_FILE
                  // Noncompliant: starts with '_'
#define _MY_FILE
#define FIELD__VAL(field) ##field // Noncompliant: contains "
int free(void *pArg, int len) { // Noncompliant: free is a s
  int __i; // Noncompliant: starts with "__"
  //...
}
#endif
```

Compliant Solution

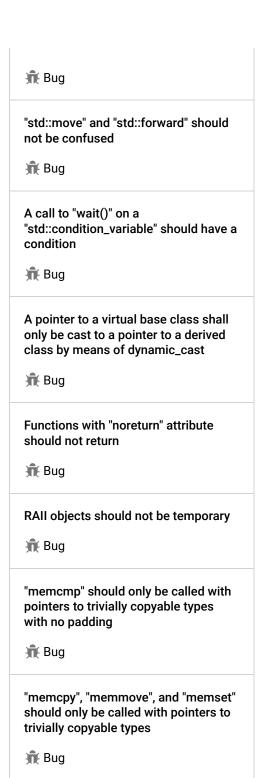
```
#ifndef MY_FILE
#define MY FILE
#define FIELD_VAL(field) ##field
int clean(void *pArg, int len) {
  int i:
  //...
#endif
```

See

- MISRA C:2004, 20.1 Reserved identifiers, macros and functions in the standard library, shall not be defined redefined or undefined.
- MISRA C++:2008, 17-0-1 Reserved identifiers, macros and functions in the standard library shall not be defined, redefined, or undefined.
- MISRA C:2012, 21.2 A reserved identifier or macro name shall not be declared
- CERT, DCL37-C. Do not declare or define a reserved identifier
- CERT, DCL51-CPP. Do not declare or define a reserved identifier

Available In:

sonarlint 😊 | sonarcloud 🙆 | sonarqube | Developer Edition



"std::auto_ptr" should not be used

Destructors should be "noexcept"

📆 Bug

📆 Bug

© 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