



Apex

**ABAP** 

C

C++

CloudFormation

COBOL

C#

**CSS** 

Flex

Go **GO** 

5 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

ΑII 578 6 Vulnerability (13) rules

**R** Bug (111)

• Security Hotspot

Tags

⊗ Code (436)

Quick 68 Fix

"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

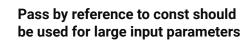
"pthread\_mutex\_t" should be unlocked in the reverse order they were locked

🖷 Bug

"pthread\_mutex\_t" should be properly initialized and destroyed

📆 Bug

"pthread\_mutex\_t" should not be consecutively locked or unlocked twice



Analyze your code

Code Smell





cppcoreguidelines performance

Search by name...

To pass an input parameter to a function, there are two possibilities: pass by value, or pass by reference to const. Which one is best depends of the size of the object, which is an indicator of the cost to copy it. A small one, with cheap copy constructors, should be passed by value, while a larger one should be passed by reference to const.

This rule detects when a parameter has been passed by value, while it should have been passed by reference to const:

- · Because it is too large
- · Because it contains virtual functions and passing it by value will slice the extra members if you happen to pass an object of a derived class.

In some cases, you may want to pass by value a large object, if you modify it in the function but you don't want the initial object to be impacted by these changes. We do not detect such a situation, which will be a false positive.

There are other ways to pass input parameters for sinks (for instance by rvalue references), but this rule is only about the choice between pass by value and pass by reference to const.

# **Noncompliant Code Example**

```
struct Student {string firstName; string lastName; Date birth
class XmlNode {
  virtual ~XmlNode();
  virtual string toString();
};
void registerStudent(School &school, Student p); // Noncompli
void dump(ostream &out, XmlNode node); // Noncompliant, XmlNo
```

# **Compliant Solution**

```
struct Student {string firstName; string lastName; Date birth
class XmlNode {
  virtual ~XmlNode();
  virtual string toString();
void registerStudent(School &school, Student const & p); // C
void dump(ostream &out, XmlNode const &node); // Compliant, n
```

# **Exceptions**

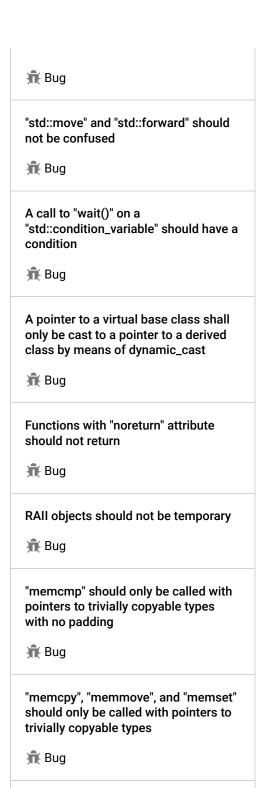
This rule does not flag large objects passed by value to coroutines because passing arguments by reference to a coroutine often leads to dangling references, e.g., after suspension and resumpion of the coroutine.

# See

• C++ Core Guidelines F.16 - For "in" parameters, pass cheaply-copied types by value and others by reference to const

Available In:

sonarlint in sonarcloud on 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