



**ABAP** 

Apex

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 ⊗ 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 twice

The order for arguments of the same type in a function call should be obvious

Analyze your code

Code Smell

Major

cppcoreguidelines bad-practice suspicious pitfall

When a function has several consecutive parameters of the same type, there is a risk that the arguments are not provided in the right order. Moreover, it is generally the sign of code which is too low-level. Maybe

- the arguments should have a stronger type
- some arguments could be grouped together to form a higher level abstraction.

The use of two parameters of the same type is useful in situations like comparing arguments, combining arguments through a binary operation and swapping arguments but three or more arguments of the same type is considered bad practice.

This rule raises an issue when a function is defined with more than two consecutive parameters of the same type. For this rule, only the "raw" type of the parameter will be considered (a string const & will be considered the same type as a std::string).

## **Noncompliant Code Example**

```
double acceleration(double initialSpeed, double finalSpeed, d
  return (finalSpeed - initialSpeed) / deltaT;
double dot_product(double x1, double y1, double x2, double y2
void f() {
  double x1, x2, y1, y2;
  auto result = dot_product(x1,x2,y1,y2);// The order is wron
  auto acc = acceleration(10, 50, 110); // Very unclear, prob
```

## **Compliant Solution**

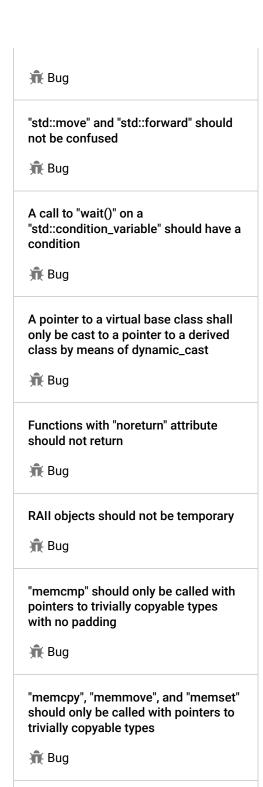
```
// This code assumes the use of a strong type / units library
Acceleration acceleration(Speed initialSpeed, Speed finalSpee
  return (finalSpeed - initialSpeed) / deltaT;
struct point {
  double x;
  double y;
double dot product(point p1, point p2);
double f() {
  point p1,p2;
  auto result = dot_product(p1,p2);
  auto acc = acceleration(50 * km / hour, 110 * km / hour, 10
```

## See

• C++ Core Guidelines I.4 - Make interfaces precisely and strongly typed

Available In:

sonarlint in 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