



SAP ABAP

APEX Apex

C C

C++

CloudFormation

COBOL COBOL

C# C#

**E** CSS

X Flex

**©** Go

5

🐇 Java

Js JavaScript

HTML

Kotlin

Kubernetes

💃 Objective C

PHP

PL/I

PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

**Terraform** 

**Text** 

Ts TypeScript

T-SQL

VB VB.NET

VB6 VB6

XML XML



## C++ static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C++ code

All 578 rules Vulnerability 13

**∄** Bug 111

Security Hotspot

ity 18 Code 436

)

O Quick 68 Fix

Tags

Search by name...

confusing duplicate suspicious

"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

<table-of-contents> 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

Rug

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

📆 Bug

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

## Methods should not have identical implementations

Analyze your code

When two methods have the same implementation, either it was a mistake - something else was intended - or the duplication was intentional, but may be confusing to maintainers. In the latter case, one implementation should invoke the

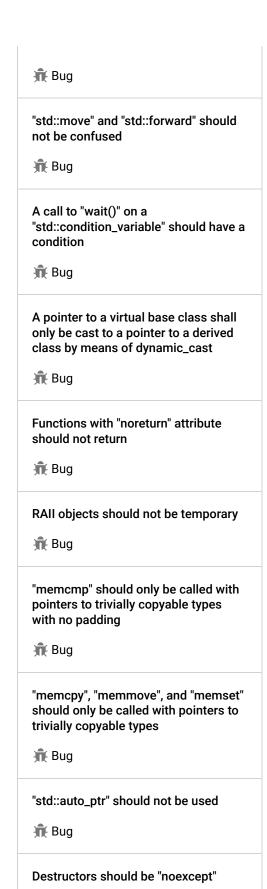
This rule raises an exception when two methods implemented inside the class definition share the same implementation.

## **Noncompliant Code Example**

```
class Point {
  int x;
  int y;
// ....
public:
  void setX(int v) {
    if (v \ge 0 \&\& v < MAX_X) {
      x = v;
      return;
    }
    error();
  void setY(int v) { // Noncompliant
    if (v \ge 0 \&\& v < MAX_X) {
      x = v;
      return;
    }
    error();
};
```

## **Compliant Solution**

```
class Point {
 int x;
 int y;
// ....
public:
void setX(int v) {
    if (v >= 0 \&\& v < MAX X) {
      x = v;
      return;
    }
    error();
  void setY(int v) {
    if (v \ge 0 \& v < MAX_X) {
     y = v;
      return;
    error();
```



📆 Bug

Exceptions

Empty methods, methods with the same name (overload) and methods with only one statement are ignored.

Available In:

sonarlint o sonarcloud o sonarqube beveloper Edition

};

© 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