



ABAP

Apex

С

C++

CloudFormation

COBOL

C#

CSS

Flex

Go =GO

HTML 5

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 578 6 Vulnerability 13 rules

R Bug (111)

o 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

Virtual functions should not have default arguments

Analyze your code

api-design pitfall

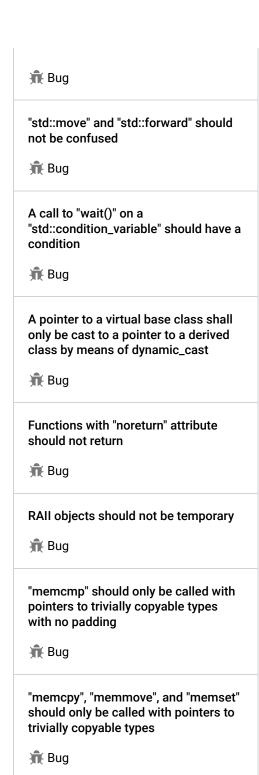
It's best to avoid giving default argument initializers to virtual functions. While doing so is legal, the code is unlikely to be correctly maintained over time and will lead to incorrect polymorphic code and unnecessary complexity in a class hierarchy.

Noncompliant Code Example

```
class Base {
public:
  virtual void fun(int p = 42) { // Noncompliant
 }
};
class Derived : public Base {
public:
  void fun(int p = 13) override { // Noncompliant
 }
};
class Derived2 : public Base {
public:
  void fun(int p) override {
    // ...
 }
};
int main() {
 Derived *d = new Derived;
  Base *b = d;
  b->fun(); // uses default argument 42
  d->fun(); // uses default argument 13; was that expected?
  Base *b2 = new Base;
  Derived2 *d2 = new Derived2;
  b2->fun(); // uses default argument 42
  d2->fun(); // compile time error; was that expected?
}
```

Compliant Solution

```
class Base {
public:
  void fun(int p = 42) { // non-virtual forwarding function
    fun_impl(p);
 }
protected:
  virtual void fun_impl(int p) {
    // ...
};
class Derived : public Base {
protected:
  void fun_impl(int p) override {
    // ...
```



"std::auto_ptr" should not be used

Destructors should be "noexcept"

📆 Bug

📆 Bug

```
class Derived2 : public Base {
protected:
   void fun_impl(int p) override {
        // ...
   }
};

See Also
   • {rule:cpp:S1712}

Available In:
sonarlint  sonarcloud  sonarqube  Developer 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