



ABAP

APEX Apex

c c

C++

CloudFormation

COBOL COBOL

C# C#

E CSS

⊠ Flex

=GO

₹ HTML

Go

🐇 Java

Js JavaScript

Kotlin

Kubernetes

Objective C

PHP

PL/I

PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

Terraform

Text

тs 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

R Bug (111)

Security Hotspot

Code 436

Quick 68 Fix

Tags

✓ Search by name...

"memset" should not be used to delete sensitive data

6 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

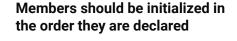
"pthread_mutex_t" should be unlocked in the reverse order they were locked

₩ 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

♥ Minor ②

Quick

?

cppcoreguidelines cert suspicious

Class members are initialized in the order in which they are declared in the class, not the order in which they appear in the class initializer list. To avoid errors caused by order-dependent initialization, the order of members in the initialization list should match the order in which members are declared in a class.

The initialization order, as described here, is:

- 1. If the constructor is for the most-derived class, virtual bases are initialized in the order in which they appear in depth-first left-to-right traversal of the base class declarations (left-to-right refers to the appearance in base-specifier lists)
- 2. Then, direct bases are initialized in left-to-right order as they appear in this class's base-specifier list
- 3. Then, non-static data members are initialized in order of declaration in the class definition.

Noncompliant Code Example

```
#include <iostream>
struct A {
    std::cout << "A(num = " << num << ")" << std::endl;
};
struct B {
  int b;
class C : public A, B {
public:
  int x;
  int y;
 C(int i) : B{i}, A{b}, y(i), x(y + 1) { } // Noncompliant
};
int main() {
 C c(1); // Undefined behavior, might print "A(num = 0)"
  std::cout << c.x << " " << c.y << std::endl; // might prin
}
```

Compliant Solution

```
#include <iostream>

struct A {
    A(int num) {
        std::cout << "A(num = " << num << ")" << std::endl;
    };

struct B {
    int b;
};

class C : public A, B {</pre>
```



"std::move" and "std::forward" should not be confused



A call to "wait()" on a "std::condition_variable" should have a condition



A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast



Functions with "noreturn" attribute should not return



RAII objects should not be temporary



"memcmp" should only be called with pointers to trivially copyable types with no padding



"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types

📆 Bug

"std::auto_ptr" should not be used

📆 Bug

Destructors should be "noexcept"

```
👬 Bug
```

```
public:
    int x;
    int y;

C(int i) : A{i}, B{i}, x(i + 1), y(i) { }
};

int main() {
    C c(1); // prints "A(num = 1)"
    std::cout << c.x << " " << c.y << std::endl; // prints "2
}</pre>
```

See

- CERT, OOP53-CPP. Write constructor member initializers in the canonical order
- <u>C++ Core Guidelines C.47</u> Define and initialize member variables in the order of member declaration

```
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