

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- 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

All rules 578

Vulnerability 13

Bug 111

Security Hotspot 18

Code Smell 436

Quick Fix 68

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, double deltaT) {
    return (finalSpeed - initialSpeed) / deltaT;
}

double dot_product(double x1, double y1, double x2, double y2) {
    return x1 * x2 + y1 * y2;
}

void f() {
    double x1,x2,y1,y2;
    auto result = dot_product(x1,x2,y1,y2); // The order is wrong
    auto acc = acceleration(10, 50, 110); // Very unclear, probably should be acceleration(10, 50, 110, 10)
}
```

### Compliant Solution

```
// This code assumes the use of a strong type / units library
Acceleration acceleration(Speed initialSpeed, Speed finalSpeed, double deltaT) {
    return (finalSpeed - initialSpeed) / deltaT;
}

struct point {
    double x;
    double y;
};

double dot_product(point p1, point p2) {
    return p1.x * p2.x + p1.y * p2.y;
}

double f() {
    point p1,p2;
    auto result = dot_product(p1,p2);
    auto acc = acceleration(50 * km / hour, 110 * km / hour, 10 * s);
}
```

### See

- [C++ Core Guidelines I.4](#) - Make interfaces precisely and strongly typed

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition

 Bug
<b>"std::move" and "std::forward" should not be confused</b>  Bug
<b>A call to "wait()" on a "std::condition_variable" should have a condition</b>  Bug
<b>A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast</b>  Bug
<b>Functions with "noreturn" attribute should not return</b>  Bug
<b>RAII objects should not be temporary</b>  Bug
<b>"memcmp" should only be called with pointers to trivially copyable types with no padding</b>  Bug
<b>"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types</b>  Bug
<b>"std::auto_ptr" should not be used</b>  Bug
<b>Destructors should be "noexcept"</b>  Bug