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

Pass by reference to const should be used for large input parameters

Analyze your code

Code Smell | Major | Quick Fix | cppcoreguidelines performance

To pass an input parameter to a function, there are two possibilities: pass by value, or pass by reference to const. Which one is best depends of the size of the object, which is an indicator of the cost to copy it. A small one, with cheap copy constructors, should be passed by value, while a larger one should be passed by reference to const.

This rule detects when a parameter has been passed by value, while it should have been passed by reference to const:

- Because it is too large
- Because it contains virtual functions and passing it by value will slice the extra members if you happen to pass an object of a derived class.

In some cases, you may want to pass by value a large object, if you modify it in the function but you don't want the initial object to be impacted by these changes. We do not detect such a situation, which will be a false positive.

There are other ways to pass input parameters for sinks (for instance by rvalue references), but this rule is only about the choice between pass by value and pass by reference to const.

Noncompliant Code Example

```
struct Student {string firstName; string lastName; Date birth
class XmlNode {
  virtual ~XmlNode();
  virtual string toString();
};
void registerStudent(School &school, Student p); // Noncompli
void dump(ostream &out, XmlNode node); // Noncompliant, XmlNo
```

Compliant Solution

```
struct Student {string firstName; string lastName; Date birth
class XmlNode {
  virtual ~XmlNode();
  virtual string toString();
};
void registerStudent(School &school, Student const & p); // C
void dump(ostream &out, XmlNode const &node); // Compliant, n
```

Exceptions

This rule does not flag large objects passed by value to coroutines because passing arguments by reference to a coroutine often leads to dangling references, e.g., after suspension and resumpion of the coroutine.

See

- C++ Core Guidelines F.16 - For "in" parameters, pass cheaply-copied types by value and others by reference to `const`

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

Bug

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

Bug

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

Bug

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

Bug

Functions with "noreturn" attribute should not return

Bug

RAII objects should not be temporary

Bug

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

Bug

"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