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

"using namespace" directives should not be used in header files

Analyze your code

Code Smell | Critical | cppcoreguidelines based-on-misra suspicious

A using directive makes names from another namespace available in the current scope. It should only be used when those names do not create an ambiguity with other names, otherwise, it is better to fully qualify the names you want to use.

When you write a header file, you don't know from which context it will be included. Therefore, if this header contains using directives, you cannot be sure that they will not create ambiguities in that context. Those ambiguities could lead to compilation failures or, worse, to a different function being selected by overload resolution depending on the order of inclusion of headers.

A using declaration behaves in the same way but only for one name. Because of their much narrower scope, this rule does not apply to using declarations.

Noncompliant Code Example

```
// f1.h
void foo ( char_t a );
namespace NS1
{
  void foo( int32_t a );
}

inline void bar ( )
{
  foo ( 0 );
}

// f2.h
namespace NS1
{
}
using namespace NS1; // Noncompliant

// f1.cc
#include "f1.h"
#include "f2.h"

int32_t m1 ( )
{
  bar ( ); // bar calls foo ( char_t );
}

// f2.cc
#include "f2.h"
#include "f1.h"
void m2 ( )
{
  bar ( ); // bar calls foo ( int32_t );
}
```

Exceptions

The issue only happens if the using directive is at global scope or at namespace scope. If is is inside a function body, it will cease to be in effect at the end of the current scope, and will not propagate to the users of the header file.

See

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

- MISRA C++:2008, 7-3-6 - using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.
- C++ Core Guidelines SF.7 - Don't write `using namespace` at global scope in a header file

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition