



ABAP

Apex Apex

C C

G C++

CloudFormation

COBOL COBOL

C# C#

**E** CSS

00.

X Flex

=GO

5 HTML

Go

🎒 Java

Js JavaScript

Kotlin

Kubernetes

Objective C

PHP

PL/I

L/SQL PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

**Terraform** 

**Text** 

Ts 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

( 18 Code

Ode (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

<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

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

Analyze your code

Code Smell

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.

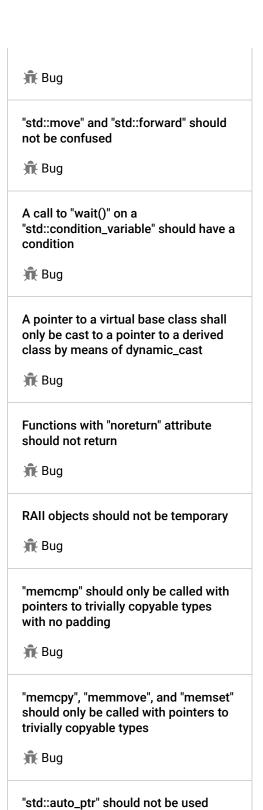
## Noncompliant Code Example

```
// 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

📆 Bug

Destructors should be "noexcept"

- 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

© 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