



ABAP

Apex

С

C++

CloudFormation

COBOL

C#

CSS

Flex

Go =GO

HTML 5

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

ΑII 578 rules

6 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

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

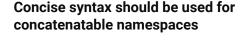
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice



Analyze your code

since-c++17 confusing

Namespaces represent a cross-file named scope. They are very useful to organize code and interfaces without cluttering a unique namespace. For instance, they provide a much cleaner way to avoid name collisions than using bad long names.

Namespaces can be nested to provide even more structure to type and symbol names. In that case, namespaces can be nested one inside another like scopes would with curly braces.

In C++17, a new concise syntax was introduced to increase the readability of nested namespaces. It is much less verbose and involves much less curly braces-delimited scopes. Whereas declaring a nested namespace of depth N requires N pairs of curly braces with the original syntax, this new syntax requires only one pair of curly braces. This syntax is much more readable and less error-prone. When possible, non-inlined or inlined (since C++20) named namespaces should be concatenated.

Noncompliant Code Example

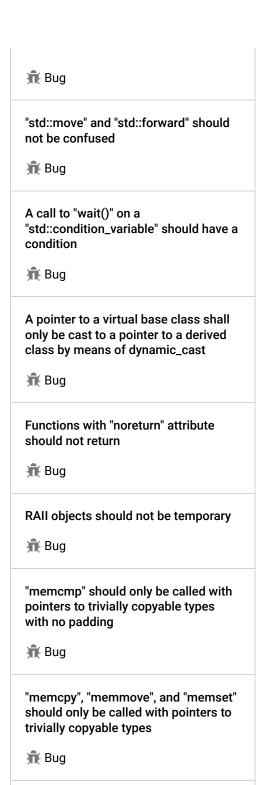
```
namespace geometry { // Noncompliant
 namespace common {
  class point {
  };
}
inline namespace triangle {
  class edge {
  };
}
```

Compliant Solution

```
namespace geometry::common {
  class point {
  };
namespace geometry::inline triangle { // C++20
  class edge {
 };
namespace sonarsource { // Compliant: cannot be concatenated
  namespace core {
    class Rule {
    };
  class A {
 };
}
```

Available In:

sonarlint in sonarcloud sonarqube Developer Edition



"std::auto_ptr" should not be used

Destructors should be "noexcept"

📆 Bug

📆 Bug

© 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