



ABAP

- Apex Apex
- **c** c
- C++
- CloudFormation
- COBOL COBOL
- C# C#
- CSS
- **⋈** Flex
- **€O** Go
- **THIML**
- Java
- Js JavaScript
- Kotlin
- Kubernetes
- **6** Objective C
- PHP
- PL/I
- 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

Tags

"memset" should not be used to delete sensitive data

Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows

Vulnerability

ective C 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

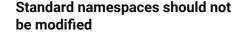
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice



Analyze your code

Search by name...

It may seem tidy to add your new declarations to the std or posix namespaces, but doing so results in undefined behavior. The C++14 Standard, [namespace.std] (ISO/IEC 14882-2014 §17.6.4.2.1), paragraphs 1 and 2 states:

- 1. The behavior of a C++ program is undefined if it adds declarations or definitions to namespace std or to a namespace within namespace std unless otherwise specified. A program may add a template specialization for any standard library template to namespace std only if the declaration depends on a user-defined type and the specialization meets the standard library requirements for the original template and is not explicitly prohibited.
- 2. The behavior of a C++ program is undefined if it declares:
 - an explicit specialization of any member function of a standard library class template, or
 - an explicit specialization of any member function template of a standard library class or class template, or
 - an explicit or partial specialization of any member class template of a standard library class or class template.

In addition to restricting extensions to the ${\tt std}$ namespace, the C++14 Standard goes on in §17.6.4.2.2 to say:

 The behavior of a C++ program is undefined if it adds declarations or definitions to namespace posix or to a namespace within namespace posix unless otherwise specified. The namespace posix is reserved for use by ISO/IEC 9945 and other POSIX standards.

You may think that it's legitimate to reopen std to define a version of extension points (std::swap, std::hash...) that work with your types, but it's not necessary: If you call these extension points according to the correct pattern (see for instance {rule:cpp:S5963} for swap), user-defined version will be found too.

This rule raises an issue for any modification of the standard std and posix namespaces.

Noncompliant Code Example

```
namespace MyNamespace {
  class MyType {/*...*/};
}
namespace std { // Noncompliant
  int x;
  void swap(MyNamespace::MyType &m1, MyNamespace::MyType &m2)
}
```

Compliant Solution

```
namespace expanded_std {
  int x;
}
namespace MyNamespace {
  class MyType {/*...*/};
  void swap(MyType &m1, MyType &m2); // See also S5963 to see
}
```

Exceptions

A namespace fragment that only contains template specializations or explicit

📆 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 Rug 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

Destructors should be "noexcept"

🕀 Bug

📆 Bug

instantiations is ignored by this rule.

See

• CERT, DCL58-CPP. - Do not modify the standard namespaces

Available In:

© 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

sonarlint ⊕ | sonarcloud ₺ | sonarqube | Developer Edition