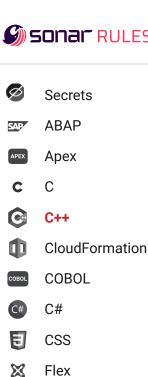
Search by name...





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

o Security ⊗ Code (436) O Quick 68 Fix All 578 6 Vulnerability (13) **R** Bug (111) Hotspot rules

Tags

"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

Immediately dangling references

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

"pthread_mutex_t" should be properly

"pthread_mutex_t" should not be consecutively locked or unlocked

initialized and destroyed

should not be created

📆 Bug

📆 Bug

📆 Bug

📆 Bug

```
Partial specialization syntax
should not be used for function
                                                Analyze your code
templates
lock-in
Class templates can be explicitly or partially specialized. But according to the C++
standard, function templates cannot be partially specialized. Under certain
conditions, the Microsoft® compiler will silently ignore the confusing application of
partial specialization syntax to a function, but other compilers will raise an error for it
and fail compilation.
Noncompliant Code Example
  template<typename T>
  void fun(T p);
  template<typename T>
  void fun<T>(T p) { // Noncompliant
  }
Compliant Solution
  template<typename T>
  void fun(T p);
  template<typename T>
  void fun(T p) {
  }
 Available In:
 sonarlint 😊 | sonarcloud 🙆 | sonarqube | Developer
```

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOLIRCE SONARI INT SONAROLIRE and SONARCLOLID 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**

I
🖟 Bug
"std::move" and "std::forward" should not be confused
∰ Bug
A call to "wait()" on a "std::condition_variable" should have a condition
n Bug
A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast
ਜ਼ਿ Bug
Functions with "noreturn" attribute should not return
👬 Bug
RAII objects should not be temporary
्रे 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
n Bug
Destructors should be "noexcept"
🖟 Bug