



ABAP

Apex

С



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 **6** Vulnerability 13 € rules

R Bug (111)

• Security Hotspot ⊗ Code (436)

Quick 68 Fix

Tags

void second();

Search by name...

since-c++11 clumsy pitfall

"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

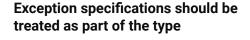
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 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

Starting C++17, exception specifications became a part of a function type. This implies that these two functions, for example, have different types:

```
void first() noexcept;
```

Making exception specifications part of the type will, for the right reason, break code where a function that throws an exception is provided in a context where noexcept function is expected.

It is important to note that the same way it is not allowed to overload based on the return type, it is also not allowed to overload based on the exception specifications.

This rule will trigger on code that will stop compiling starting C++17, and on explicit casts that add noexcept to a type.

Noncompliant Code Example

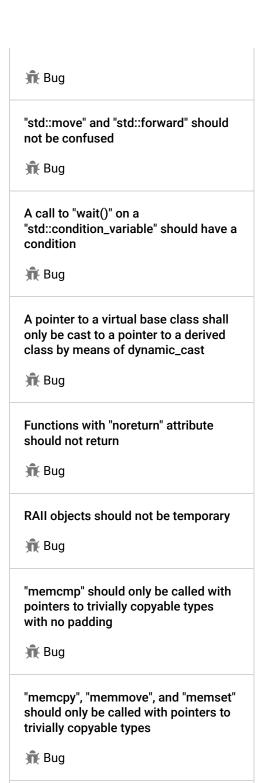
```
template<typename T>
void callF1(T t1, T t2) {
  t1();
  t2();
void f1();
void f1NoExcept() noexcept;
int main() {
  callF1(f1, f1NoExcept); // Noncompliant, f1 and f1NoExcept
  std::function<void() noexcept> fptr1 = f1; // Noncompliant
  void (*fptr2)() noexcept = f1; // Noncompliant
  void (*fptr3)() noexcept = (void (*)() noexcept) f1; // Non
}
```

Compliant Solution

```
template<typename T1, typename T2>
void callF1(T1 t1, T2 t2) {
  t2();
void f1();
void f1NoExcept() noexcept;
int main() {
  callF1(f1, f1NoExcept); // Compliant
  std::function<void() noexcept> fptr1 = f1NoExcept; // Compl
  void (*fptr2)() = f1; // Compliant
```

Available In:

sonarlint 😊 | sonarcloud 🙆 | sonarqube | Develop 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