Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
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

All rules 578 | Vulnerability 13 | Bug 111 | Security Hotspot 18 | Code Smell 436 | Quick Fix 68

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

Exception specifications should be treated as part of the type

Analyze your code

Code Smell | Major | since-c++11 clumsy pitfall

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;
void second();
```

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) {
  t1();
  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 Developer Edition

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

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

Bug

Destructors should be "noexcept"

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