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

Assignment operators should not be "virtual"

Analyze your code

Code Smell | Blocker | cppcoreguidelines pitfall

C++ does not support polymorphic copy or move assignment operators. For example, the signature of a copy assignment operator on a "Base" class would be `Base& operator=(const Base& other)`.

And on a "Derived" class that extends "Base", it would be `Derived& operator=(const Derived& other)`.

Because these are two entirely different method signatures, the second method does not override the first, and adding `virtual` to the "Base" signature does not change which method is called.

It is possible to add an `operator=` override in a derived class, but doing so is an indication that you may need to reexamine your application architecture.

Noncompliant Code Example

```
class Base {
public:
  virtual Base& operator=(const Base& other); // Noncompliant
};

class Derived : public Base {
public:
  Derived& operator=(const Derived& other);
};
```

Compliant Solution

```
class Base {
protected:
  Base& operator=(const Base& other); // not virtual
};

class Derived : public Base {
public:
  Derived& operator=(const Derived& other);
};
```

See

- C++ Core Guidelines C.60 - Make copy assignment non-virtual, take the parameter by const&, and return by non-const&
- C++ Core Guidelines C.63 - Make move assignment non-virtual, take the parameter by &&, and return by non-const &

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

initialized and destroyed

Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Bug

"std::move" and "std::forward" should not be confused

Bug

A call to "wait()" on a "std::condition_variable" should have a

© 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