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

Inherited functions should not be hidden

Analyze your code

Code Smell | Critical | cppcoreguidelines | confusing

An inherited member function can be hidden in a derived class and that creates a class that behaves differently depending on which interface is used to manipulate it.

Overriding happens when the inherited method is virtual and a method declared in the derived class uses the same identifier as well as the same signature (the return types can be different, as long as they are covariant). However, if the inherited method is non-virtual or if the two declarations of the method do not share the same signature, the method of the base class will be hidden.

Such a class increases the inheritance complexity, and confuses consumers with its non-polymorphic behavior, which can lead to errors.

Noncompliant Code Example

```
class Base {
public:
  void shutdown();
  virtual void log(int a);
};

class Derived : public Base {
public:
  void shutdown(); //Noncompliant
  void log(float a); //Noncompliant
};

void stopServer(Base *obj, Derived *obj2) {
  obj->shutdown(); // always calls Base::shutdown even if the
  obj->log(2); // calls Base::log(int) even if the given obje
  obj2->shutdown(); // calls Derived::shutdown
  obj2->log(2); // calls Derived::log(float), even if this re
}
```

Compliant Solution

```
class Base {
public:
  void shutdown();
  virtual void log(int a);
};

class Derived : public Base {
public:
  void shutdownAndUpdate(); // Define a method with a differe
  void log(int a) override; // Or make the method a proper ov
};

void stopServer(Base *obj) {
  obj->shutdown(); // calls Base::shutdown and there is no co
  obj->log(2); // calls Derived::log(int) if the given object
}
```

See

- C++ Core Guidelines C.138 - Create an overload set for a derived class and its bases with using

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

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition