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

Child class fields should not shadow parent class fields

Analyze your code

Code Smell | Blocker | cppcoreguidelines | confusing

Having a variable with the same name in two unrelated classes is fine, but do the same thing within a class hierarchy and you'll get confusion at best, chaos at worst.

Noncompliant Code Example

```
class Fruit {
  protected:
    Season ripe;
    static Color flesh;

  // ...
};

class Raspberry : public Fruit {
  private:
    bool ripe;  // Noncompliant
    static Color FLESH; // Noncompliant
};
```

Compliant Solution

```
class Fruit {
  protected:
    Season ripe;
    static Color flesh;

  // ...
};

class Raspberry : public Fruit {
  private:
    bool ripened;
    static Color FLESH_COLOR;

};
```

Exceptions

This rule ignores same-name fields that are `static` in both the parent and child classes. This rule ignores `private` parent class fields, but in all other such cases, the child class field should be renamed.

```
class Fruit {
  private:
    Season ripe;
  // ...
};

class Raspberry : public Fruit {
```

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

```
  private:
    Season ripe;  // Compliant as parent field 'ripe' is anyw
  // ...
};
```

or

```
class Fruit {
  public:
    Season ripe;
  // ...
};

class RedFruit : private Fruit {
};

class Raspberry : public RedFruit { // RedFruit inherits from
  private:
    Season ripe;  // Compliant as parent field 'ripe' is anyw
  // ...
};
```

See

- C++ Core Guidelines - ES.12 - Do not reuse names in nested scopes

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition