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

"std::initializer_list" constructor should not overlap with other constructors

Analyze your code

Code Smell | Minor | confusing since-c++11

When a class has a constructor accepting `initializer_list` of type `X` and another constructor that has n parameters of either type `X` or a type that can be converted to `X`, the constructor call resolution becomes complex. This makes code hard to reason about and might lead to calls resolving to unexpected constructors. What makes it even more complex, is that the constructor resolution rules are different if `X` is a type template parameter.

This rule flags classes that have constructors overlapping with the `initializer_list` constructor. It is recommended to simplify the class by:

- A technical change: replace `initializer_list` parameter by a `std::vector`, a `std::array`, or a variadic template. This way the caller is forced to be more explicit.
- A design change: make the construction of an object of type `X` taking object(s) of type `Y` as parameters equivalent to constructing it with an initializer list containing the object(s) of type `Y`. This way you can reduce the number of overlapping constructors to the one that takes `initializer_list`.

Noncompliant Code Example

```
class A { // Noncompliant
public:
  A();
  A(int); // This constructor  overlaps with the initializer_
  A(int, long); // This constructor overlaps with the initial
  A(std::initializer_list<int>); // "initializer_list" constr
  A(int, A);
  A(int, double);
};

void f1() {
  A a1(10); // A(int) is called
  A a2{10}; // The "initializer_list" constructor is called
  A a3(10, 1l); // A(int, long) is called
  A a4{10, 1l}; // The "initializer_list" constructor is call
  A a5{10, A{}}; // A(int, A) is called
  // A a6{10, 1.2}; // doesn't compile
}

class B { // Noncompliant
public:
  B(int); // This constructor overlaps with the initializer_l
  B(int, long); // This constructor doesn't overlap with the
  template<typename T>
  B(std::initializer_list<T>); // "initializer_list" construc
};

void f2() {
  B b1(10); // The constructor with single "int" parameter is
  B b2{10}; // The "initializer_list" constructor is called
  B b3(10, 1l); // B(int, long) is called
  B b4{10, 1l}; // B(int, long) is called
}
```

Compliant Solution

```
class A {
```

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

```
public:
  A();
  A(int);
  A(int, long);
  A(std::vector<int>);
  A(int, A);
  A(int, double);
};

void f1() {
  A a1(10); // A(int) is called
  A a2{10}; // A(int) is called
  A a3(10, 1l); // A(int, long) is called
  A a4{10, 1l}; // A(int, long)
  A a5{10, A{}}; // A(int, A) is called
  A a6{10, 1.2}; // A(int, A) is called
  A a7 {{1,2,4}}; // vector is called no confusion
}

class B {
public:
  B(int, long);
  template<typename T>
  B(std::initializer_list<T>);
};

void f2() {
  B b1({10}); // The "initializer_list" constructor is called
  B b2{10}; // The "initializer_list" constructor is called
  B b3(10, 1l); // B(int, long) is called
  B b4{10, 1l}; // B(int, long) is called
}
```

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition