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

"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

Empty class members should be marked as "[[no_unique_address]]"

Analyze your code

Code Smell   Minor   since-c++20 performance

In C++ every independent object needs to have a unique address, which implies that its size cannot be null. Sub-objects of another object, however, do not have this constraint. Empty base class subobjects usually don't take any space in the final object, but empty member variables, by default, take some space, at least one byte. The impact of the final size of the object may be even larger, due to padding and alignment requirements.

C++ 20 introduces the `[[no_unique_address]]` attribute. It indicates that preserving the uniqueness of address guarantee is not important for the decorated member variable, and if the variable type is empty, no storage needs to be reserved for it in the class.

If the type is not empty, this attribute is still valid, and has no effect. This allows to place this attribute on dependant member variables in template classes, and have the exact behavior depend on the actual template parameters.

This rule raises an issue on each member of a class that has an empty or potentially empty (in case of templates) type and does not have `[[no_unique_address]]` attribute.

Note: This rule is disabled on Windows because `[[no_unique_address]]` isn't well supported by MSVC and Clang on this platform.

Noncompliant Code Example

```
struct Empty {};
struct Wrapped {
  int* ptr;
  Empty e; // Noncompliant
}; // sizeof(Wrapped) is > sizeof(int*)

template<typename K, typename V, typename Hash, typename Equa
class HashMap {
  /* ... */
  Hash hash; // Noncompliant if HashMap is instantiated with
  Equal equal; // Noncompliant if HashMap is instantiated wit
};
```

Compliant Solution

```
struct Empty {};
struct Wrapped {
  int* ptr;
  [[no_unique_address]] Empty e;
}; // sizeof(Wrapped) can be equal to sizeof(int*)

template<typename K, typename V, typename Hash, typename Equa
class HashMap {
  /* ... */
  [[no_unique_address]] Hash hash;
  [[no_unique_address]] Equal equal;
};
```

Exceptions

This rule does not apply to fields whose class has a non-default alignment.

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