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

Comparison operators should not be virtual

Analyze your code

Code Smell | Blocker | cppcoreguidelines pitfall

Making a comparison operator `virtual` implies that you want to compare objects of different types by overriding `operator==`, for instance, in a subclass to compare instances of the base class with instances of the subclass. But polymorphic comparison operators are very difficult to get right, and are actually questionable in concept. After all, can two objects with only a few common members really be equal?

This rule raises issues on `virtual` comparison operators.

Noncompliant Code Example

```
struct Foo {
  virtual bool operator==(const Foo &other) const; // Noncomp
  virtual bool operator!=(const Foo &other) const; // Noncomp
};
```

Compliant Solution

```
struct Foo {
  bool operator==(const Foo &other) const;
  bool operator!=(const Foo &other) const;
};
```

See

- C++ Core Guidelines C.87 - Beware of == on base classes

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