

- 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

"shared_ptr" should not be taken by rvalue reference

Analyze your code

Code Smell

Minor ?

Quick Fix ?

cppcoreguidelines confusing since-c++11 clumsy

Taking a `shared_ptr` by r-value reference is unnecessary. If done on purpose, it might imply that `unique_ptr` is a better choice since it transfers unique ownership.

In general, a function should take:

- A copy of a `shared_ptr`, if the function takes part in the ownership of the managed object
- A reference to a `shared_ptr` if the function plans to modify the `shared_ptr` itself by calling `reset` or `swap`
- A reference to a `const shared_ptr`, if the function might take part in the ownership by copying the reference to another `shared_ptr` on at least one of its paths
- A raw pointer/reference to the object, if the function is only interested in the current value of the managed object

Noncompliant Code Example

```
Class Circle{};

void fn(shared_ptr<Circle>&& circle); // Noncompliant
```

Compliant Solution

```
Class Circle{};

void fn(shared_ptr<Circle> circle);
```

Exceptions

The move constructor of the `shared_ptr` itself.

See

- [C++ Core Guidelines R.34](#) - Warn if a function takes a `Shared_ptr<T>` by rvalue reference. Suggesting taking it by value instead.

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition

 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