

- 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 rules578
- Vulnerability13
- Bug111
- Security Hotspot18
- Code Smell436
- Quick Fix68

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

## "reinterpret\_cast" should not be used

Analyze your code

Code Smell

Major

cppcoreguidelines pitfall

Because `reinterpret_cast` does not perform any type safety validations, it is capable of performing dangerous conversions between unrelated types.

Since C++20, a `std::bit_cast` should be used instead of `reinterpret_cast` to reinterpret a value as being of a different type of the same length preserving its binary representation, as the behavior of `reinterpret_cast` is undefined in such case.

This rule raises an issue when `reinterpret_cast` is used.

### Noncompliant Code Example

```
class A { public: virtual ~A(){} };
class B : public A { public: void doSomething(){} };

void func(A *a, float f) {
    if (B* b = reinterpret_cast<B*>(a)) { // Noncompliant
        b->doSomething();
    }
    int x = *reinterpret_cast<int*>(f); // Noncompliant
}
```

### Compliant Solution

```
class A { public: virtual ~A(){} };
class B : public A { public: void doSomething(){} };

void func(A *a, float f) {
    if (B* b = dynamic_cast<B*>(a)) {
        b->doSomething();
    }
    int x = std::bit_cast<int>(f);
}
```

### See

- CppCoreGuidelines, Type safety profile - Type.1: Don't use reinterpret\_cast.

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

 Bug
<b>"std::move" and "std::forward" should not be confused</b>  Bug
<b>A call to "wait()" on a "std::condition_variable" should have a condition</b>  Bug
<b>A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast</b>  Bug
<b>Functions with "noreturn" attribute should not return</b>  Bug
<b>RAII objects should not be temporary</b>  Bug
<b>"memcmp" should only be called with pointers to trivially copyable types with no padding</b>  Bug
<b>"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types</b>  Bug
<b>"std::auto_ptr" should not be used</b>  Bug
<b>Destructors should be "noexcept"</b>  Bug