


-  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

C-style and functional notation casts should not be used

Analyze your code

 Code Smell  Major  based-on-misra pitfall

C++ allows the traditional C-style casts [E.G. `(int) f`] and functional notation casts [E.G. `int(f)`], but adds its own forms:

- `static_cast<type>(expression)`
- `const_cast<type>(expression)`
- `dynamic_cast<type>(expression)`
- `reinterpret_cast<type>(expression)`
- `std::bit_cast<type>(expression)` (since C++20)

C-style casts and functional notation casts are largely functionally equivalent. However, when they do not invoke a converting constructor, C-style casts are capable of performing dangerous conversions between unrelated types and of changing a variable's `const`-ness. Attempt to do these things with an explicit C++-style cast, and the compiler will catch the error. Use a C-style or functional notation cast, and it cannot.

Moreover, C++20 has introduced a `std::bit_cast` as a way of reinterpreting a value as being of a different type of the same length preserving its binary representation. The behavior of such conversion when performed via C-style cast or `reinterpret_cast` is undefined.

Additionally, C++-style casts are preferred because they are visually striking. The visual subtlety of a C-style or functional cast may mask that a cast has taken place, but a C++-style cast draws attention to itself, and makes the the programmer's intention explicit.

This rule raises an issue when C-style cast or functional notation cast is used.

Noncompliant Code Example

```
#include <iostream>

class Base { };

class Derived: public Base
{
public:
    int a;
};

void DoSomethingElse(Derived *ptr)
{
    ptr->a = 42;
}

void DoSomething(const Base *ptr)
{
    Derived* derived = (Derived*)ptr; // Noncompliant; inadvert
    DoSomethingElse(derived);
}

void checksBits(float f)
{
    int x = *(int*)&f; // Noncompliant; has undefined behavior
}

int main(int argc, char* argv[])
{
    Derived *ptr = new Derived();
    ptr->a = 1337;
```

 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

```
DoSomething(ptr);

std::cout << ptr->a << std::endl; /* 1337 was expected, but

return 0;
}
```

Compliant Solution

```
/* ... */

void DoSomething(const Base *ptr)
{
    /* error: static_cast from type 'const Base*' to type 'Derived*'
    Derived* derived = static_cast<Derived*>(ptr); // Compliant
    DoSomethingElse(derived);
}

void checksBits(float f)
{
    int x = std::bit_cast<int>(f);
}

/* ... */
```

Exceptions

Void casts and explicit constructor calls are allowed.

See

- MISRA C++:2008, 5-2-4 - C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used.

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition