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

"dynamic_cast" should be used for downcasting

Analyze your code

Code Smell   Minor   cppcoreguidelines suspicious

Casting a base-class pointer/reference to a derived-class pointer/reference is commonly referred to as downcasting which can only be done using an explicit cast.

However, the use of `static_cast` for such a cast is unsafe because it doesn't do any runtime check. If the cast memory doesn't contain an object of the expected derived type, your program enters the undefined behavior territory.

If your object is polymorphic, you might prefer using `dynamic_cast` instead, as it allows safe downcasting by performing a run-time check:

- If the cast memory contains an object of the expected derived type, the check succeeds. The result of the `dynamic_cast` points/refers to the derived object.
- If the cast memory doesn't contain an object of the expected derived type, the check fails. If the `dynamic_cast` is used on a pointer, `nullptr` is returned. If it was used on a reference, `std::bad_cast` is thrown.

This rule raises an issue when `static_cast` is used for downcasting.

Noncompliant Code Example

```
struct Shape {
  virtual ~Shape();
  // ...
};

struct Rectangle : public Shape {
  double width;
  double height;
};

struct Circle : public Shape {
  double radius;
};

double computeArea(const Shape* shape) {
  const auto* rectangle = static_cast<const Rectangle*>(shape
  return rectangle->width * rectangle->height;
}
```

Compliant Solution

```
struct Shape {
  virtual ~Shape();
  // ...
};

struct Rectangle : public Shape {
  double width;
  double height;
};

struct Circle : public Shape {
  int radius;
};

double computeArea(const Shape* shape) {
  if(const auto* rectangle = dynamic_cast<const Rectangle*>(s
    return rectangle->width * rectangle->height;
```

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

```
  }
  return 0;
}
```

See

- C++ Core Guidelines - Type safety profile - Type.2: Don't use static_cast to downcast. Use dynamic_cast instead.

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition