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

Destructors should be "noexcept"

Analyze your code

Bug | Blocker | cppcoreguidelines error-handling since-c++11

Throwing an exception from a destructor results in undefined behavior, meaning that your program could be terminated abruptly without neatly destroying others objects.

Thus destructors should never `throw` exceptions. Instead, they should catch and handle those thrown by the functions they call, and be `noexcept`.

This rule raises an issue when a destructor is not `noexcept`. By default, destructors are `noexcept`, therefore most of the time, nothing needs to be written in the source code. A destructor is not `noexcept` if:

- the base class or a data member has a non `noexcept` destructor,
- the destructor is decorated with the `noexcept` keyword followed by something that evaluates to false.

Noncompliant Code Example

```
struct A {
  ~A() noexcept(false) {} // Noncompliant
};

struct C {
  /* ... */
  A a; // This member data prevents automatic declaration of
  ~C() { // Noncompliant
    /* ... */
  }
};
```

Compliant Solution

```
struct A {
  ~A() noexcept(true) {}
};

struct C {
  /* ... */
  A a;
  ~C() { // Compliant, noexcept by default
    /* ... */
  }
};
```

See

- C++ Core Guidelines C.36 - A destructor may not fail
- C++ Core Guidelines C.37 - Make destructors noexcept

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