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

Move and swap operations should be "noexcept"

Analyze your code

Code Smell | Blocker | cppcoreguidelines error-handling since-c++11

Move operations (move constructor, move assignment operator) are all about efficient resource stealing. When stealing resources from the source, you don't have to allocate any memory or perform any other operation that might fail. This is why most people will expect move operation to be non-throwing.

Additionally, if a move operation fails, the source object can have been partially altered by the move, making recovery very tricky, or just impossible. Therefore, to ensure robustness, some functions (for instance, `std::move_if_noexcept`, used by `std::vector`) will decide to copy your object if its move operations are not decorated with `noexcept`. This can significantly slow down your program.

If you can not implement your move operations so that they never throw, you may as well only provide copy operations that will be safer to use.

Swap operations are very similar to move operations, in that they should be equivalent to moving two objects into each other. So if you are adding a swap function to your type, it should be *noexcept* too.

Note that for most classes, you should not write your own move operations, but rely on the "Rule-of-Zero" ({rule:cpp:S4963}).

This rule raises an issue when a move or swap operation is not *noexcept*, which can happen in two cases:

- The operation is user-defined, and is not unconditionally declared as `noexcept`,
- The operation is implicitly defined, and one of the base classes or member variables of the class does not have `noexcept` move operations.

Noncompliant Code Example

```
struct A {
  A (A const &a);
  A (A && a); // Noncompliant
  ~A();
  A &operator=(A const &a);
  A &operator=(A &&a); // Noncompliant
};

void swap(A& a1, A& a2); // Noncompliant
```

Compliant Solution

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

```
struct A {
  A (A const &a);
  A (A && a) noexcept;
  ~A();
  A &operator=(A const &a);
  A &operator=(A &&a) noexcept;
};

void swap(A& a1, A& a2) noexcept;
```

See

- C++ Core Guidelines C.66 - Make move operations noexcept
- C++ Core Guidelines C.85 - Make swap operations `noexcept`

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
Privacy Policy

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition