

- 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

Exceptions should not be used

Analyze your code

Code Smell Major ?

While exceptions are a common feature of modern languages, there are several reasons to potentially avoid them:

- They make the control flow of a program difficult to understand, because they introduce additional exit points.
- The use of exceptions in new code can make that code difficult to integrate with existing, non-exception-safe code.
- They add to the size of each binary produced, thereby increasing both compile time and final executable size.
- They may incur a small performance penalty.
- The time required to handle an exception is not easy to assess, which makes them difficult to use for hard real-time applications.

This rule raises an issue when:

- an exception is thrown
- a try-catch block is used
- an exception specification (`throw (xxx)`) is present.

Noncompliant Code Example

This C++ code example also applies to Objective-C.

```
double myfunction(char param) throw (int); // Noncompliant
void f {
    try // Noncompliant
    {
        do_something();
        throw 1; // Noncompliant
    }
    catch (...)
    {
        // handle exception
    }
}
```

Compliant Solution

```
double myfunction(char param) noexcept;
bool f {
    if (!do_something()); {
        // Handle the situation
        return false;
    }
    // Rest of the code
    return true;
}
```

Exceptions

`noexcept` specifications are ignored, because even if you choose not to use exceptions in your code, it's important to decorate as `noexcept` certain functions (for instance, move constructors that do not `throw`). This decoration can be detected by type traits, and some meta-programming techniques rely on this information.

 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

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition

© 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](#)