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

"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

RAII objects should not be temporary

Analyze your code

Bug Blocker cppcoreguidelines suspicious

The RAII idiom associates the lifetime of a resource with the lifetime of an object: The resource is acquired when the object is created, and released when it is destroyed.

If the object that controls the resource lifetime is a temporary, chances are it will get destroyed while the resource should still be in use, leading to resource corruption. This rule detects temporaries that look like RAII objects.

Noncompliant Code Example

```
void f() {
  scoped_lock{myMutex}; // Noncompliant. The mutex will be lo
  protectedCode(); // This code is not protected by the mutex
}
```

Compliant Solution

```
void f() {
  scoped_lock lock{myMutex}; // Compliant
  protectedCode();
  // The mutex is correctly released at this point
}
```

See

- C++ Core Guidelines ES.84 - Don't (try to) declare a local variable with no name

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

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