sonar RULES

Products ⌄

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

---

The addresses of standard library functions should not be taken   **Analyze your code**

🐞 Code Smell   🔥 Critical ❓   🏷 pitfall

Taking the address of a library function is not something robust: The library might make changes to a function that are compatible with a normal use of a function, but not with taking its address (for instance, adding a parameter with a default value, or adding an overload to an overload set). More specifically, the standard library has stated that there would be no barrier against such changes, and that for stability users should not take the address of standard library functions.

Starting with C++20, it's no longer allowed to take the address of a standard library function (with some exceptions with functions for formatting streams).

Noncompliant Code Example

```
int main() {
  std::unique_ptr<std::FILE, int(*)(std::FILE*)> fp(
    std::fopen("test.txt", "r"),
    std::fclose); // Noncompliant, address of fclose is impli
  // Work with fp
}
```

Compliant Solution

```
int main() {
  std::unique_ptr<std::FILE, int(*)(std::FILE*)> fp(
    std::fopen("test.txt", "r"),
    [](std::FILE*file){return std::fclose(file);});
  // Work with fp
}
```

See

- Document of the C++ standardization committee about change management in the standard library.

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

🐛 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