sonar RULES Products

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

Access specifiers should not be redundant

Analyze your code

Code Smell Minor redundant clumsy

Redundant access specifiers should be removed because they needlessly clutter the code.

Noncompliant Code Example

```
struct S {
  public: // Noncompliant; does not affect any declaration
  private:
    void method();
  private: // Noncompliant; does not change accessibility lev
    int member;
  private: // Noncompliant; does not affect any declaration
};
class C {
    int member;
  private: // Noncompliant;  does not change accessibility le
    void method();
};
```

Compliant Solution

```
struct S {
  private:
    void method();
    int member;
};
class C {
    int member;
    void method();
};
```

Exceptions

An access specifier at the very beginning of a `class` or `struct` that matches the default access level is ignored even when it doesn't change any accessibility levels.

```
class C {
  private: // redundant but accepted
    // ...
};
struct S {
  public: // redundant but accepted
    // ...
};
```

Such a specifier is redundant, but ignored to allow `class`es and `struct`s to be described uniformly.

```
class C {
  public:
    void call();

  protected:
    int delete();
```

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

```
  private:
    int code;
};
struct S {
  public: // redundant but accepted
    int sum();

  protected:
    int min();

  private:
    int count;
};
```

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