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

"std::auto_ptr" should not be used

Analyze your code

Bug | Blocker | suspicious since-c++11

`std::auto_ptr` was a pre-C++11 attempt to do what `std::unique_ptr` now does. Unfortunately, the move semantics needed to make it work properly weren't in place, so copying a `std::auto_ptr` has the very surprising behavior of invalidating the source of the copy.

That problem has been fixed with `std::unique_ptr`, so `std::auto_ptr` has been deprecated in C++11 and removed in C++17.

If your compiler allows it, you should replace all use of `std::auto_ptr` with `std::unique_ptr`. Otherwise, define your own (non-copyable) smart pointer.

Noncompliant Code Example

```
using namespace std;

void draw(auto_ptr<Shape> p) { cout << s->x() << ", " << s.y(

void f()
{
    std::auto_ptr<Shape> s = createShape(); // Noncompliant
    draw(s); // This call invalidates s
    draw(s); // This call will crash, because s is null
}
```

Compliant Solution

```
using namespace std;

void draw(unique_ptr<Shape> p) { cout << s->x() << ", " << s.

void f()
{
    std::unique_ptr<Shape> s = createShape();
    // draw(s); // Would not compile
    draw(move(s)); // Will compile, and the user knows s has
}
```

Available In:
sonarlint sonarcloud sonarqube Developer Edition

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