rules.sonarsource.com

C++ static code analysis: Pointers or references obtained from aliased smart pointers should not be used as function parameters

2 minutes

One way to end up with a dangling pointer or a dangling reference is to pass to a function a pointer or a reference which lifecycle is not controlled. This is the case when the pointer or the reference is obtained from a smart pointer (`shared_ptr` or `unique_ptr`) which is not locally defined or which is potentially aliased.

In this case, nothing can guarantee the developer that the pointer or the reference coming from this smart pointer will always be valid: for example, this smart pointer could be reset somewhere in the call chain.

Noncompliant Code Example

```
class Shape;

std::shared_ptr<Shape> globalShape;

void g(Shape& s) {
  globalShape.reset();
  // do something with parameter s
}

void f1() {
  g(*globalShape); // Noncompliant, lifecycle of the reference is not controlled, parameter s of function g will be a dangling reference
}
```

Compliant Solution

```
class Shape;

std::shared_ptr<Shape> globalShape;

void g(Shape& s) {
  globalShape.reset();
  // do something with parameter s
}

void f1() {
  auto myShape = globalShape; // reference count of the smart
pointer is incremented, the pointer-to object is kept alive
  g(*myShape );
}
```

See

- C++ Core Guidelines R.37 - Do not pass a pointer or reference obtained from an aliased smart pointer