rules.sonarsource.com

# C++ static code analysis: Default capture should not be used

2 minutes

---

Lambdas can use variables from their enclosing scope (called "capture") either by reference or by value. Since lambdas may run asynchronously, reference capture should be used with caution because by the time the lambda runs, the referenced variable may be out of scope, resulting in an access violation at run time.

You can specify default capture by reference (`[&]`), or by value (`[=]`). Clearly default reference capture can cause scope issues, but default value capture can also lead to problems. That's because both forms of default capture implicitly also capture `*this`, which would automatically be used if for example you referenced a method from the enclosing scope.

If the lambda is used immediately (for instance, passed as an argument to `std::sort`), there is no risk of dangling reference. For those lambdas, it is safe to pass everything through a default capture by reference. See also {rule:cpp:S5495}.

This rule raises an issue when default capture is used, unless the lambda is immediately executed.

## Noncompliant Code Example

```
void fun() {
  Foo foo;
  ...
  executor->Schedule([&] {  // Noncompliant
    maybeMember(foo);  // implicit use of *this reference if
maybeMember is a member function. foo and maybeMember may
both be gone by the time this is invoked
  });
}
```

## Compliant Solution

```
void fun() {
  Foo foo;
  ...
  executor->Schedule([&foo] { // it is clear that foo is captured by
reference and compilation is going to fail if maybeMember is a
member function
    maybeMember(foo);
  });
}
```