rules.sonarsource.com

C++ static code analysis: "bind" should not be used

3 minutes

Using `std::bind` or `boost::bind` allows to create a wrapper to a function where some arguments are bound to fixed values, or the order of arguments is changed. However, this way of wrapping a function comes with some issues:

- The code is not really clear, the usage of placeholders makes it difficult to follow,
- The complexity of the implementation of `std::bind` often results in sub-optimal code,
- The possibilities offered by `bind` are limited, for instance, it's not possible to bind a function with variable arguments,
- `bind` allows to silently discard some arguments, which is often not what was expected,
- Fixed arguments are evaluated when `bind` is called, not when the bound function is called, which might be surprising in some cases,
- `bind` requires to take the address of a function, which can be difficult (if the function is overloaded) or not recommended (for standard library functions, see {rule:cpp:S5180}).

To create such a wrapper, it is usually better to:

- either write a simple lambda that just calls the wrapped function with the full power of the language concerning how the arguments are going to be tinkered with.
- or, if the function `bind` is only applied to the first arguments, call `std::bind_front` that has been introduced in C++20 and that is safer to use and a simpler replacement than lambdas.

Noncompliant Code Example

```
constexpr int multiply(int a, int b) {return a*b;}
auto flipMultiplication = std::bind(multiply, std::placeholders::_2,
std::placeholders::_1); // Noncompliant
int i = flipMultiplication (6, 7);

struct Logger {
   string subsystem;
   template <class T>
   void operator()(T const &what) {  cout << subsystem << ": " <<
what << "\n"; }
};

void f(){
   Logger net {"network"};
   // Noncompliant in C++14, compliant by exception in C++11
because it would require a polymorphic lambda
   auto networkLog = bind(net, _1);
   networkLog(4);
}
```

Compliant Solution

```
constexpr int multiply(int a, int b) {return a*b;}
auto flipMultiplication = [](int a, int b) {return multiply(b, a);}; //
Compliant
int i2 = flipMultiplication (6, 7);

void f(){
   Logger net {"network"};
   auto networkLog = [&](auto what) {net(what);}; // Compliant
   networkLog(4);
}{code}
```

Exceptions

Before C++14 lambda was limited and could not capture by move or be polymorphic. Therefore, if the call to `bind` makes use of

those features, it will not be reported in C++11.

See

- Abseil - Avoid std::bind
- Effective modern C++ item 34: Prefer lambdas to `std::bind`