rules.sonarsource.com

C++ static code analysis: "std::string_view" and "std::span" parameters should be directly constructed from sequences

2-3 minutes

`std::string_view` (introduced in C++17) and `std::span` (introduced in C++20) are thin generic wrappers for a contiguous sequence of elements. These wrappers can be used to unify the interface of functions that were previously accepting references to specific container types: `const std::string&`, `const std::vector<int>&`...

One of the benefits of such modernization is that it eliminates the need to explicitly create a temporary container. This happens in situations where part of the sequence is passed as an argument: `substr` is called on `std::string`. It can also happen when the type of the container elements needs to be adjusted: converting `std::vector<T*>` to `std::vector<const T*>`. When changing the type of a function parameter to `std::string_view` or `std::span` the modification of the function call site to remove the no longer needed temporary might be missed and the code will still compile. This rule will help eliminate these temporaries.

This rule raises an issue when an unnecessary temporary is passed as an argument to a parameter of `std::string_view` or `std::span` type.

Noncompliant Code Example

```
void parse(std::string_view sv);
bool oddAre0(std::span<int const* const> nums);
std::vector<int*> getNums();

void caller(std::string const& s) {
  parse(s.substr(10)); // Noncompliant
  parse(std::string(s, 2, 5)); // Noncompliant
  parse(std::string(s.data(), 20)); // Noncompliant
  parse(std::string(s.data(), 10)); // Noncompliant

  std::vector<int*> nums = getNums();
  if (oddAre0(std::vector<int const*>{nums.begin(), nums.end()})) { //
Noncompliant: This copy is verbose and slow
    // ...
```

```
  }
}
```

Compliant Solution

```
void parse(std::string_view sv);
bool oddAre0(std::span<int const* const> nums);
std::vector<int*> getNums();

void caller(std::string const& s) {
  parse(std::string_view(s).substr(10));
  parse(std::string_view(s).substr(2, 5));
  parse(std::string_view(s.data(), 20));
  parse({ s.data(), 10 });

  std::vector<int*> nums = getNums();
  if (oddAre0(nums)) {
      // ...
  }
}
```

See

- RSPEC-6009 - using `std::string_view` as a parameter type
- RSPEC-6188 - using `std::span` as a parameter type