rules.sonarsource.com

C++ static code analysis: "std::cmp_*" functions should be used to compare unsigned values with negative values

2 minutes

Comparison between `signed` and `unsigned` integers is dangerous because it produces counterintuitive results outside of their common range of values.

When a signed integer is compared to an unsigned one, the former might be converted to unsigned. The conversion preserves the two's-complement bit pattern of the signed value that often corresponds to a large unsigned result. For example, `2U < -1` is `true`.

C++20 introduced remedy to this common pitfall: a family of `std::cmp_*` functions defined in the `<utility>` header. These functions correctly handle negative numbers and lossy integer conversion. For example, `std::cmp_less(2U, -1)` is `false`.

This rule raises an issue when an unsigned integer is compared with a negative value.

Noncompliant Code Example

```
bool less = 2U < -1; // Noncompliant
```

```
unsigned x = 1;
signed y = -1;
if (x < y) { // Noncompliant
  // ...
}
```

Compliant Solution

```
bool less = std::cmp_less(2U, -1); // Compliant
```

```
unsigned x = 1;
signed y = -1;
if (std::cmp_less(x, y)) { // Compliant
  // ...
}

bool fun(int x, std::vector<int> const& v) {
  return std::cmp_less(x, v.size()); // Compliant
}
```

```
std::vector<int> v = foo();
if (0 < v.size() && v.size() < 100) { // Compliant, even
though v.size() returns an unsigned integer
}
```

See

- {rule:cpp:S845} - a more generic rule about mixing signed and unsigned values.

- {rule:cpp:S6183} - a version of this rule that triggers as soon as signed and unsigned variables are compared.