- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- Kubernetes
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TypeScript
- T-SQL
- VB.NET
- VB6
- XML

C++ static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C++ code

All rules 578 | Vulnerability 13 | Bug 111 | Security Hotspot 18 | Code Smell 436 | Quick Fix 68

Tags

Search by name...

"memset" should not be used to delete sensitive data
Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows
Vulnerability

XML parsers should not be vulnerable to XXE attacks
Vulnerability

Function-like macros should not be invoked without all of their arguments
Bug

The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist
Bug

Assigning to an optional should directly target the optional
Bug

Result of the standard remove algorithms should not be ignored
Bug

"std::scoped_lock" should be created with constructor arguments
Bug

Objects should not be sliced
Bug

Immediately dangling references should not be created
Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked
Bug

"pthread_mutex_t" should be properly initialized and destroyed
Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

"std::cmp_*" functions should be used to compare signed and unsigned values

Analyze your code

Code Smell | Minor | since-c++20 symbolic-execution bad-practice pitfall

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 starts by detecting comparisons between signed and unsigned integers. Then, if the signed value can be proven to be negative, the rule {rule:cpp:S6214} will raise an issue (it is a bug). Otherwise, this rule will raise an issue. Therefore, if this rule is enabled, {rule:cpp:S6214} should be enabled too.

Noncompliant Code Example

```
bool less = 2U < -1; // Compliant, raises S6214

bool foo(unsigned x, signed y) {
  return x < y; // Noncompliant: y might be negative
}

bool fun(int x, std::vector<int> const& v) {
  return x < v.size(); // Noncompliant: x might be negative
}
```

Compliant Solution

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

bool foo(unsigned x, signed y) {
  return std::cmp_less(x, y); // Compliant
}

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

void compute(std::vector<int> const &v) {
  if (0 < v.size() && v.size() < 100) { // Compliant, even th
  }
}
```

See

- {rule:cpp:S845} - a more generic rule about mixing signed and unsigned values.
- {rule:cpp:S6214} - a version of this rule that only triggers when it detects negative values are involved.

Bug

"std::move" and "std::forward" should not be confused

Bug

A call to "wait()" on a "std::condition_variable" should have a condition

Bug

A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast

Bug

Functions with "noreturn" attribute should not return

Bug

RAII objects should not be temporary

Bug

"memcmp" should only be called with pointers to trivially copyable types with no padding

Bug

"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types

Bug

"std::auto_ptr" should not be used

Bug

Destructors should be "noexcept"

Bug

Available In:

sonarlint sonarcloud sonarqube Developer Edition