- 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

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

Function parameters should not be of type "std::unique_ptr<T> const &"

Analyze your code

Code Smell   Major   cppcoreguidelines design bad-practice since-c++11 clumsy

If you use `std::unique_ptr<T> const &` for a function parameter type, it means that the function will not be able to alter the ownership of the pointed-to object by the `unique_ptr`:

- It cannot acquire ownership of the pointed-to object (this would require a parameter of type `std::unique_ptr<T>`)
- It cannot transfer the object ownership to someone else (this would require a `std::unique_ptr<T> &`).

That means the function can only observe the pointed-to object, and in this case passing a `T*` (if the `unique_ptr` can be null) or a `T&` (if it cannot) provides the same features, while also allowing the function to work with objects that are not handled by a `unique_ptr` (E.G. objects on the stack, in a `vector`, or in another kind of smart pointer), thus making the function more general-purpose.

Noncompliant Code Example

```
using namespace std;
void draw(unique_ptr<Shape> const &shape); // Noncompliant

void drawAll(vector<unique_ptr<Shape>> v)
{
  for (auto &shape : v) {
      if (shape) {
        draw(shape);
      }
  }
}
```

Compliant Solution

```
using namespace std;
void draw(Shape const &shape); // Compliant

void drawAll(vector<unique_ptr<Shape>> v)
{
  for (auto &shape : v) {
      if (shape) {
        draw(*shape);
      }
  }
}
```

See

- C++ Core Guidelines R.32 - Take a unique_ptr<widget> parameter to express that a function assumes ownership of a widget

Available In:

sonarlint sonarcloud sonarqube Developer Edition