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

Virtual functions should not have default arguments

Analyze your code

Code Smell   Critical   api-design pitfall

It's best to avoid giving default argument initializers to virtual functions. While doing so is legal, the code is unlikely to be correctly maintained over time and will lead to incorrect polymorphic code and unnecessary complexity in a class hierarchy.

Noncompliant Code Example

```
class Base {
public:
  virtual void fun(int p = 42) { // Noncompliant
    // ...
  }
};

class Derived : public Base {
public:
  void fun(int p = 13) override { // Noncompliant
    // ...
  }
};

class Derived2 : public Base {
public:
  void fun(int p) override {
    // ...
  }
};

int main() {
  Derived *d = new Derived;
  Base *b = d;
  b->fun(); // uses default argument 42
  d->fun(); // uses default argument 13; was that expected?

  Base *b2 = new Base;
  Derived2 *d2 = new Derived2;
  b2->fun(); // uses default argument 42
  d2->fun(); // compile time error; was that expected?
}
```

Compliant Solution

```
class Base {
public:
  void fun(int p = 42) { // non-virtual forwarding function
    fun_impl(p);
  }
protected:
  virtual void fun_impl(int p) {
    // ...
  }
};

class Derived : public Base {
protected:
  void fun_impl(int p) override {
    // ...
  }
```

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

```
};

class Derived2 : public Base {
protected:
  void fun_impl(int p) override {
    // ...
  }
};
```

See Also

- {rule:cpp:S1712}

Available In:

sonarlint sonarcloud sonarqube Developer Edition