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

Member variables should be initialized

Analyze your code

Bug Critical symbolic-execution bad-practice pitfall

In a C++ class, all member variables of a type with an implicit or explicit constructor will be automatically initialized. This is not the case for the others: if no initialization is explicitly written, the variable will be left uninitialized.

This comes with all the risks associated with uninitialized variables, and these risks propagate to all the classes using the faulty class as a type. This is all the more surprising that most programmers expect a constructor to correctly initialize the members of its class (this is its raison d'être after all).

To avoid such situations, all non class type fields should always be initialized (in order of preference):

- With an in-class initializer
- In the initialization list of a constructor
- In the constructor body

See {rule:cpp:S3230} for more details about this order.

Noncompliant Code Example

```
class C {
  int val = 42;
};

class S {
public:
  C c;
  int i;
  int j;

  S() : i(0) {} // Noncompliant: this->j is left uninitialize
  S(bool) : i(0), j(0) {}
};

class T {
public:
  S s;

  T() : s() {} // Noncompliant: s.j is left uninitialized
  T(bool b) : s(b) {}
};

class U {
public:
  T t;

  U() : t() {} // Noncompliant: t.s.j is left uninitialized
};
```

Compliant Solution

```
class C {
  int val = 42;
};

class S_fixed {
public:
  C c;
  int i;
```

```
  int j;

  S_fixed() : i(0), j(0) {} // Compliant
  S_fixed(bool) : i(0), j(0) {}
};

class T_fixed {
public:
  S _fixed s;

  T_fixed() : s() {}  // Compliant
  T_fixed(bool b) : s(b) {}
};

class U {
public:
  T t;

  U() : t() { t.s.j = 0; }  // Compliant
};
```

Exceptions

Aggregate classes do not initialize most of their data members, but allow their users to use nice and flexible initialization syntax. They will be ignored by this rule (but are the subject of {rule:cpp:S5558}).

See

- C++ Core Guidelines C.41: A constructor should create a fully initialized object

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

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