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

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

Bug | Blocker | unpredictable

The function `memcmp` can only be used for objects of trivially copyable types. This includes scalar types, arrays, and trivially copyable classes.

A class type is trivially copyable if:

- One or more of the following methods is trivial and the rest are deleted: copy constructor, move constructor, copy assignment operator, and move assignment operator,
- It has a trivial, non-deleted destructor.

Additionally, if the type contains padding, some of its bits might be non-representative, and a strict comparison of raw memory contents might lead to the mistaken belief that two identical objects are actually different.

Noncompliant Code Example

```
class Shape { // Trivially copyable, but will contain padding
public:
  bool visible;
  int x;
  int y;
};

bool isSame(Shape *s1, Shape *s2)
{
    return memcmp(s1, s2, sizeof Shape) == 0; // Noncompliant
}
```

Compliant Solution

```
class Shape {
public:
  bool visible;
  int x;
  int y;
};

bool operator==(Shape const &s1, Shape const &s2) {
  return s1.visible == s2.visible && s1.x == s2.x && s1.y ==
}

bool isSame(Shape *s1, Shape *s2)
{
    return (*s1) == (*s2);
}
```

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