



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

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Analyze your code

Bug Blocker symbolic-execution multi-threading

Mutexes are synchronization primitives that allow to manage concurrency.

- non recursive mutexes* are targeted by this rule. They can be locked/unlocked only once. Any locking/unlocking sequence that contains two consecutive identical operations leads to an undefined behaviour.
- recursive mutexes* are not target by this rule. They can be locked several times and unlocked several times as long as the number of locks/unlocks is the same.

This rule raises an issue when a `pthread_mutex_t` is locked or unlocked several times in a row. We assume that all `pthread_mutex_t` are non-recursive (this is the most common case).

Noncompliant Code Example

```
pthread_mutex_t mtx1;

void bad1(void)
{
    pthread_mutex_lock(&mtx1);
    pthread_mutex_lock(&mtx1);
}

void bad2(void)
{
    pthread_mutex_unlock(&mtx1);
    pthread_mutex_unlock(&mtx1);
}
```

Compliant Solution

```
pthread_mutex_t mtx1;

void ok(void)
{
    pthread_mutex_lock(&mtx1);
    pthread_mutex_unlock(&mtx1);
}
```

See

- [The Open Group](#) pthread_mutex_init, pthread_mutex_destroy

Available in:

sonarlint sonarcloud sonarqube Developer Edition

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