



## 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

### Use "std::variant" instead of unions with non-trivial types.

Analyze your code

Code Smell Blocker since-c++17 clumsy pitfall

In order to save memory, unions allow you to use the same memory to store objects from a list of possible types as long as one object is stored at a time.

In C and in C++ prior to C++11, unions are restricted to trivial types.

Starting from C++11, it is possible to use unions with non-trivial types with the following limitations :

- You have to manually handle the lifetime of the active member, using placement new and explicit object destruction.
- You have to define special members like destructor and copy-constructor while taking into consideration the active member.

In some cases, code that fails to perfectly follow those rules may still compile, but lead to memory corruption.

C++17 introduced `std::variant` which can replace unions while removing this burden and the associated risk. As a safer and more readable alternative, they should be preferred.

#### Noncompliant Code Example

```
#include <new> // Required for placement 'new'.
#include <string>
#include <iostream>

using namespace std;

struct IntOrString {
    enum {holdsInt, holdsString} currentAlternative;
    union {
        int z;
        string s; // Noncompliant: non-trivial type in Union
    };
    IntOrString() : currentAlternative{holdsInt} {
        z = 0;
    }
    IntOrString(char const *s) : currentAlternative{holdsString} {
        new(&s) string(s);
    }
    IntOrString(IntOrString const &src) : currentAlternative{src.currentAlternative} {
        if (currentAlternative == holdsString) {
            new(&s) string(src.s);
        }
    }
    IntOrString &operator=(IntOrString &&) = delete;
    ~IntOrString() {
        if (currentAlternative == holdsString) {
            s.~string();
        }
    }
};

void stringize(IntOrString &ios) {
```

<p>initialized and destroyed</p> <p> Bug</p>	<pre>if (ios.currentAlternative == IntOrString::holdsString) {     return; } new (&amp;ios.s) string(std::to_string(ios.z)); }</pre> <pre>int main() {     IntOrString ios;     auto copy = ios;     ios.z = 12;     stringize(ios);     std::cout&lt;&lt; ios.s &lt;&lt; "\n"; }</pre>
<p>"pthread_mutex_t" should not be consecutively locked or unlocked twice</p> <p> Bug</p>	
<p>"std::move" and "std::forward" should not be confused</p> <p> Bug</p>	
<p>A call to "wait()" on a "std::condition_variable" should have a</p>	<p><b>Compliant Solution</b></p> <pre>#include &lt;variant&gt; #include &lt;iostream&gt; #include &lt;string&gt;  using namespace std; using IntOrString = variant&lt;int, string&gt;;  void stringize(IntOrString &amp;ios) {     if(auto i = get_if&lt;int&gt;(&amp;ios)) {         ios = to_string(*i);     } }  int main() {     IntOrString ios = 12;     auto copy = ios;     stringize(ios);     cout &lt;&lt; std::get&lt;string&gt;(ios) &lt;&lt; '\n'; }</pre> <p>Available In:</p> <div>   Developer Edition</div>