

- 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

"switch" statements should cover all cases

Analyze your code

Code Smell Major ? suspicious

For completeness, a `switch` over the values of an `enum` must either address each value in the `enum` or contain a default case. `switch` statements that are not over `enum` must end with a default case.

This rule is a more nuanced version of {rule:cpp:S131}. Use {rule:cpp:S131} if you want to require a default case for every `switch` even if it already handles all enumerators of an `enum`. Otherwise, use this rule.

### Noncompliant Code Example

```
typedef enum {APPLE, GRAPE, KIWI} fruit;

void example(fruit f, int i) {
    switch (f) { // Noncompliant; no case for KIWI
        case APPLE:
            //...
        case GRAPE:
            //...
        case 3: // Noncompliant; case value not in enum
            // ...
    }

    switch (i) { // Noncompliant; no default
        case 0:
            // ...
        case 1:
            // ...
    }
}
```

### Compliant Solution

```
typedef enum {APPLE, GRAPE, KIWI} fruit;

void example(fruit f) {
    switch (f) {
        case APPLE:
            //...
        case GRAPE:
            //...
        default:
            // ...
    }

    switch (i) {
        case 0:
            // ...
        case 1:
            // ...
        default:
            // ...
    }
}
```

or

```
typedef enum {APPLE, GRAPE, KIWI} fruit;
```

 Bug
<b>"std::move" and "std::forward" should not be confused</b>  Bug
<b>A call to "wait()" on a "std::condition_variable" should have a condition</b>  Bug
<b>A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast</b>  Bug
<b>Functions with "noreturn" attribute should not return</b>  Bug
<b>RAII objects should not be temporary</b>  Bug
<b>"memcmp" should only be called with pointers to trivially copyable types with no padding</b>  Bug
<b>"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types</b>  Bug
<b>"std::auto_ptr" should not be used</b>  Bug
<b>Destructors should be "noexcept"</b>  Bug

```
void example(fruit f) {
    switch (f) {
        case APPLE:
            //...
        case GRAPE:
            //...
        case KIWI:
            //...
    }

    switch (i) {
        case 0:
        case 1:
            // ...
        default:
            // ...
    }
}
```

See

- [C++ Core Guidelines - Enum.2](#) - Use enumerations to represent sets of related named constants

See Also

- {rule:cpp:S131}

Available In:

sonarlint

sonarcloud

sonarqube Developer Edition