sonar RULES

Products

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 have "default" clauses

Analyze your code

Code Smell   Critical   cwe based-on-misra cert

The requirement for a final `default` clause is defensive programming. The clause should either take appropriate action, or contain a suitable comment as to why no action is taken. When the `switch` covers all current values of an `enum` - and especially when it doesn't - a `default` case should still be used because there is no guarantee that the `enum` won't be extended.

Note that there is a more nuanced version of this rule: {rule:cpp:S3562}. Use this rule if you want to require a `default` case for every `switch` even if it already handles all enumerators of an `enum`. Otherwise, use {rule:cpp:S3562}.

Noncompliant Code Example

```
switch (param) { // Noncompliant - default clause is missing
  case 0:
    doSomething();
    break;
  case 1:
    doSomethingElse();
    break;
}
```

Compliant Solution

```
switch (param) {
  case 0:
    doSomething();
    break;
  case 1:
    doSomethingElse();
    break;
  default:
    doDefault();
    break;
}
```

See

- MISRA C:2004, 15.0 - The MISRA C *switch* syntax shall be used.
- MISRA C:2004, 15.3 - The final clause of a switch statement shall be the default clause
- MISRA C++:2008, 6-4-3 - A switch statement shall be a well-formed switch statement.
- MISRA C++:2008, 6-4-6 - The final clause of a switch statement shall be the default-clause
- MISRA C:2012, 16.1 - All switch statements shall be well-formed
- MISRA C:2012, 16.4 - Every *switch* statement shall have a *default* label
- MISRA C:2012, 16.5 - A *default* label shall appear as either the first or the last *switch label* of a *switch* statement
- MITRE, CWE-478 - Missing Default Case in Switch Statement
- CERT, MSC01-C. - Strive for logical completeness

See Also

- {rule:cpp:S3562}

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

Available In:

sonarlint sonarcloud sonarqube Developer Edition