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

Control should not be transferred into a complex logic block using a "goto" or a "switch" statement

Analyze your code

Code Smell Blocker lock-in cert misra-c++2008 pitfall

Having a `switch` and its cases wholly encompassed by a control structure such as a `try`, `@try`, `catch`, `@catch`, or a loop is perfectly acceptable. (`try` and `catch` are used hereafter to refer to both variants.) It is also acceptable to have a `goto` and its target label wholly encompassed in a control structure.

What is not acceptable is using a `goto` or `case` to suddenly jump into the body of a `try`, `catch`, Objective-C `@finally`, or loop structure. Tangling labels or `switch` blocks with other control structures results in code that is difficult, if not impossible to understand. More importantly, when it compiles (some of these constructs won't compile under ISO-conformant compilers), it can lead to unexpected results. Therefore this usage should be strictly avoided.

This C++ code sample, which is also applicable to Objective-C if `try` and `catch` are converted to `@try` and `@catch`, demonstrates jumping into a `switch` and into a `try` and `catch` :

Noncompliant Code Example

```
void f ( int32_t i )
{
  if ( 10 == i )
  {
    goto Label_10; // Noncompliant; goto transfers control in
  }

  if ( 11 == i )
  {
    goto Label_11; // Noncompliant; goto transfers control in
  }

  switch ( i )
  {
    case 1:
      try
      {
        Label_10:
        case 2:  // Noncompliant; switch transfers control in
          // Action
          break;
      }
      catch ( ... )
      {
        Label_11:
        case 3: // Noncompliant; switch transfers control int
          // Action
          break;
      }
      break;
    default:
    {
      // Default Action
      break;
```

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

```
    }
  }
}
```

Compliant Solution

```
void f ( int32_t i )
{
  switch ( i )
  {
    case 1:
    case 2:
      // Action
      break;
    case 3:
      // Action
      break;
    case 10:

    default:
    {
      // Default Action
      break;
    }
  }

  try
  {
    if ( 2 == i || 10 == i)
    {
      // Action
    }
  }
  catch ( ... )
  {
    if (3 == i || 11 == i)
    {
      // Action
    }
  }
}
```

See

- MISRA C++:2008, 15-0-3 - Control shall not be transferred into a try or catch block using goto or switch statement
- CERT, MSC20-C. - Do not use a switch statement to transfer control into a complex block

Available In:

sonarlint sonarcloud sonarqube Developer Edition