sonar RULES

Products

- 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

"goto" statements should not be used to jump into blocks

Analyze your code

Code Smell | Blocker | based-on-misra brain-overload pitfall

Use of `goto` can lead to programs that are extremely difficult to comprehend and analyse, and possibly to unspecified behavior.

Unfortunately, removing `goto` from some code can lead to a rewritten version that is even more difficult to understand than the original. Therefore, limited use of `goto` is sometimes advised.

However, the use of `goto` to jump into or out of a sub-block of code, such as into the body of a `for` loop is never acceptable, because it is extremely difficult to understand and will likely yield results other than what is intended.

Noncompliant Code Example

```
void f1 (int a) {
  if (a <=0) {
    goto L2;  // Noncompliant; jumps into a different block
  }

  if (a == 0) {
  {
    goto L1; // Compliant
  }
  goto L2;  // Noncompliant; jumps into a block

L1:
  for (int i = 0; i < a; i++) {
  L2:
    //...  Should only have come here with a >=0. Loop is inf
  }
}
```

Compliant Solution

```
void f1 (int a) {
  if (a <=0) {
    // ...
  }

  if (a == 0) {
  {
    goto L1; // Compliant
  }

L1:
  for (int i = 0; i < a; i++) {
  L2:
    //...
  }
}
```

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

See

- MISRA C++:2008, 6-6-1 - Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement
- MISRA C:2012, 15.3 - Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement

Available In:

sonarlint sonarcloud sonarqube Developer Edition