

- Secrets
- Apex

ABAP

- С
- C++
- CloudFormation
- COBOL
- C#
- CSS

Flex

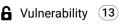
- Go =GO
- HTML 5
- Java
- **JavaScript**
- Kotlin
- Kubernetes
- Objective C
- PHP
- PL/I
- PL/SQL
- Python
- **RPG**
- Ruby
- Scala
- Swift
- Terraform
- Text
- **TypeScript**
- T-SQL
- **VB.NET**
- VB6
- **XML**



C++ static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C++ code

ΑII 578 rules



R Bug (111)

o Security Hotspot

⊗ Code (436)

O Quick 68 Fix

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

Variables should not be shadowed

Analyze your code

based-on-misra cert suspicious pitfall

Overriding or shadowing a variable declared in an outer scope can strongly impact the readability, and therefore the maintainability, of a piece of code. Further, it could lead maintainers to introduce bugs because they think they're using one variable but are really using another.

Noncompliant Code Example

```
class Foo
public:
  void doSomething();
private:
  int myField;
};
void Foo::doSomething()
    int myField = 0; // Noncompliant
}
```

```
void f(int x, bool b) {
 int y = 4;
 if (b) {
    int x = 7; // Noncompliant
    int y = 9; // Noncompliant
    // ...
}
```

Compliant Solution

```
class Foo
public:
  void doSomething();
private:
 int myField;
};
void Foo::doSomething()
    int myInternalField = 0; // Compliant
    // ...
}
```

```
void f(int x, bool b) {
 int y = 4;
 if (b) {
   int z = 7; // Better yet: Use meaningful names
   int w = 9;
    // ...
```



"std::move" and "std::forward" should not be confused



A call to "wait()" on a "std::condition_variable" should have a condition



A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast



Functions with "noreturn" attribute should not return



RAII objects should not be temporary



"memcmp" should only be called with pointers to trivially copyable types with no padding



"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types



"std::auto_ptr" should not be used

📆 Bug

Destructors should be "noexcept"

```
📆 Bug
```

Exceptions

It is common in a constructor to have constructor arguments shadowing the fields that they will initialize. This pattern avoids the need to select new names for the constructor arguments, and will not be reported by this rule:

```
class Point{
public:
 Point(int x, int y) : x(x), y(y) {} // Compliant by excepti
private:
 int x;
 int y;
};
```

See

- MISRA C:2004, 5.2 Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier
- MISRA C++:2008, 2-10-2 Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope
- MISRA C:2012, 5.3 An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
- CERT, DCL01-C. Do not reuse variable names in subscopes
- CERT, DCL51-J. Do not shadow or obscure identifiers in subscopes

Available In:

sonarlint 😊 | sonarcloud 🙆 | sonarqube | Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved. **Privacy Policy**