C static code analysis

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

All rules 311 | Vulnerability 13 | Bug 74 | Security Hotspot 18 | Code Smell 206 | Quick Fix 14

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

"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
Bug

Functions with "noreturn" attribute should not return
Bug

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

A cast shall not remove any const or volatile qualification from the type of a pointer or reference

Analyze your code

Code Smell | Critical | cppcoreguidelines cert misra-c++2008 misra-c2004 suspicious misra-c2012

Using `const` in your code improves reliability and maintenance. When passing a `const` value, developers assume that its value won't be changed. But using `const_cast<>()` to cast away a `const` qualifier, destroys developer assumptions and code reliability. It is a bad practice and reveals a flaw in the design. Furthermore, it may have an undefined behavior.

Noncompliant Code Example

```
User& func(const int& value, const User& user) {
  const_cast<int&>(value) = 2; // Noncompliant and undefined
  return const_cast<User&>(user); // Noncompliant
}
```

Compliant Solution

```
User& func(int& value, User& user) {
  value = 2;
  return user;
}
```

See

- MISRA C:2004, 11.5 - A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer
- MISRA C++:2008, 5-2-5 - A cast shall not remove any const or volatile qualification from the type of a pointer or reference
- MISRA C:2012, 11.8 - A cast shall not remove any const or volatile qualification from the type pointed to by a pointer
- CERT, EXP32-C. - Do not access a volatile object through a nonvolatile reference
- CERT, EXP05-C. - Do not cast away a const qualification
- CERT, EXP55-CPP. - Do not access a cv-qualified object through a cv-unqualified type
- C++ Core Guidelines Type.3 - Don't use const_cast to cast away const (i.e., at all): Don't cast away const.

Available In:

sonarlint sonarcloud sonarqube Developer Edition

Stack allocated memory and non-owned memory should not be freed

Bug

Closed resources should not be accessed

Bug

Dynamically allocated memory should be released

Bug

Freed memory should not be used