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 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

Keywords introduced in later specifications should not be used as identifiers

Analyze your code

Code Smell   Blocker   lock-in

While keywords introduced in later standards can legally be used as identifiers in code compiled to earlier standards, doing so will eventually cause problems. Such code will cause compile errors if (when) the compiler is upgraded, and fixing those errors could be difficult and painful.

Additionally, such misuse of keywords has the potential to thoroughly confuse people who are unfamiliar with the code base, possibly leading them to introduce additional errors.

For these reasons, the earlier this practice is stopped, the better.

This rule flags instances of the following keywords used as identifiers:

C99

`inline`, `restrict`, `_Bool`, `_Complex`, `_Noreturn`, `_Static_assert`, `_Thread_local`

C11

`_Alignas`, `_Alignof`, `_Atomic`, `_Generic`, `_Imaginary`

C++11

`alignas`, `alignof`, `char16_t`, `char32_t`, `constexpr`, `decltype`, `noexcept`, `nullptr`, `static_assert`, `thread_local`

C++20

`concept`, `requires`, `constinit`, `consteval`, `co_await`, `co_return`, `co_yield`, `char8_t`

Noncompliant Code Example

```
int inline = 0;
```

Compliant Solution

```
int inline_count = 0;
```

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