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

Header guards should be followed by according "#define" macro

Analyze your code

Code Smell | Critical | unpredictable

Include guards, wrapping around the entire content of a header file, are a best practice that ensure that no matter how many times the header is actually included in a translation unit, its content will only be seen once.

The include guard pattern is made up of four parts:

- `#ifndef` at the top of the file, with a unique macro name (usually the name relates to the name of the file, to ensure uniqueness).
- `#define` with the same macro name.
- The content of the file
- `#endif` at the end of the file

The rule raises an issue when the name in the second part differs from the name in the first (usually because of a typo or a copy/paste issue).

Noncompliant Code Example

```
#ifndef MYFILE_H
#define MY_FILE_H // Noncompliant
//...
#endif
```

Compliant Solution

```
#ifndef MYFILE_H
#define MYFILE_H
//...
#endif
```

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