- ⊘ Secrets
- SAP ABAP
- APEX Apex
- C **C**
- C++
- CloudFormation
- COBOL COBOL
- C# C#
- CSS
- Flex
- GO Go
- HTML
- Java
- JS JavaScript
- Kotlin
- Kubernetes
- Objective C
- php PHP
- PL/I PL/I
- PL/SQL PL/SQL
- Python
- RPG RPG
- Ruby
- Scala
- Swift
- Terraform
- Text
- TS TypeScript
- T-SQL
- VB VB.NET
- VB6 VB6
- XML 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

Unary prefix operators should not be repeated

Analyze your code

🐛 Bug   ⊘ Major ?

The needless repetition of an operator is usually a typo. After all, why write `!!!i` when `!i` will do?

On the other hand, the repetition of increment and decrement operators may have been done on purpose, but doing so obfuscates the meaning, and should be simplified.

This rule raises an issue for sequences of: `!`, `~`, `–`, and `+`, and in C++ for repetitions of the increment and decrement operators.

Noncompliant Code Example

```
int i = 1;

int j = – – –i;  // Noncompliant; just use -i
int k = ~~i;     // Noncompliant; same as i
int m = + +i;    // Noncompliant; operators are useless here

bool b = false;
bool c = !!!b;   // Noncompliant
```

Compliant Solution

```
int i =  1;

int j = -i;
int k =  i;
int m =  i;

bool b = false;
bool c = !b;
```

Exceptions

Boolean normalization `!!` is ignored.

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