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

Limited dependence should be placed on operator precedence

Analyze your code

⚙ Code Smell   ⬥ Major ?   🏷 cwe cert

The rules of operator precedence are complicated and can lead to errors. For this reason, parentheses should be used for clarification in complex statements. However, this does not mean that parentheses should be gratuitously added around every operation.

Parentheses are not needed:

- with a unary operator, except when ! is used as left operand in comparison expressions
- when all the operators in an expression are the same
- when only a single operator is involved
- around the right-hand side of an assignment operator unless the right-hand side itself contains an assignment

Parentheses are needed:

- in the condition of a ternary operator if it uses operators
- when overloaded shift operator << or >> is used in an expression with comparison operators

Noncompliant Code Example

```
x = a + b;
x = a * -1;
x = a + b + c;
x = f ( a + b, c );

x = a == b ? a : a - b; // Noncompliant
x = a + b - c + d; // Noncompliant
x = a * 3 + c + d; // Noncompliant

if (a = f(b,c) == true) { ... } // Noncompliant; == evaluated
x - b ? a : c; // Noncompliant; "-" evaluated first
s << 5 == 1; // Noncompliant; "<<" evaluated first
```

Compliant Solution

```
x = a + b;
x = a * -1;
x = a + b + c;
x = f ( a + b, c );

x = ( a == b ) ? a : ( a - b );
x = ( a + b ) - ( c + d );
x = ( a * 3 ) + c + d;

if ( (a = f(b,c)) == true ) { ... }
(x - b) ? a : c; // Compliant
(s << 5) == 1; // Compliant
```

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

See

- MISRA C:2004, 12.1 - Limited dependence should be placed on C's operator precedence rules in expressions
- MISRA C:2004, 12.2 - The value of an expression shall be the same under any order of evaluation that the standard permits.
- MISRA C:2004, 12.5 - The operands of a logical && or || shall be primary-expressions.
- MISRA C++:2008, 5-0-1 - The value of an expression shall be the same under any order of evaluation that the standard permits.
- MISRA C++:2008, 5-0-2 - Limited dependence should be placed on C++ operator precedence rules in expressions
- MISRA C++:2008, 5-2-1 - Each operand of a logical && or || shall be a postfix-expression.
- MISRA C:2012, 12.1 - The precedence of operators within expressions should be made explicit
- CERT, EXP00-C. - Use parentheses for precedence of operation
- CERT, EXP53-J. - Use parentheses for precedence of operation
- MITRE, CWE-783 - Operator Precedence Logic Error

Available In:

sonarlint 😊 | sonarcloud 🔵 | sonarqube 〰 Developer Edition