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

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

Braces should be used to indicate and match the structure in the non-zero initialization of arrays and structures

Analyze your code

Code Smell Major based-on-misra

ISO/IEC 14882:2003 [1] requires initializer lists for arrays, structures and union types to be enclosed in a single pair of braces (though the behaviour if this is not done is undefined). The rule given here goes further in requiring the use of additional braces to indicate nested structures.

This forces the developer to explicitly consider and demonstrate the order in which elements of complex data types are initialized (e.g. multi-dimensional arrays).

The zero initialization of arrays or structures shall only be applied at the top level.

The non-zero initialization of arrays or structures requires an explicit initializer for each element.

A similar principle applies to structures, and nested combinations of structures, arrays and other types.

Note also that all the elements of arrays or structures can be initialized (to zero or NULL) by giving an explicit initializer for the first element only. If this method of initialization is chosen then the first element should be initialized to zero (or NULL), and nested braces need not be used.

Noncompliant Code Example

```
int a1[3][2] = { 1, 2, 3, 4, 5, 6 }; // Noncompliant
int a2[5] = { 1, 2, 3 }; // Noncompliant, partial initializat
int a3[2][2] = { { }, { 1, 2 } }; // Noncompliant, zero initi
```

Compliant Solution

```
int a1[3][2] = { { 1, 2 }, { 3, 4 }, { 5, 6 } }; // Compliant
int a2[5] = { 1, 2, 3, 0, 0 }; // Compliant, Non-zero initial
int a2[5] = { 0 }; // Compliant, zero initialization
int a3[2][2] = { }; // Compliant, zero initialization
```

See

- MISRA C:2004, 9.2 - Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.
- MISRA C++:2008, 8-5-2 - Braces shall be used to indicate and match the structure in the nonzero initialization of arrays and structures.
- MISRA C:2012, 9.2 - The initializer of an aggregate or union shall be enclosed in braces.
- MISRA C:2012, 9.3 - Arrays shall not be partially initialized.

Available In:

sonarlint sonarcloud sonarqube Developer Edition