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

"#pragma pack" should be used correctly

Analyze your code

Bug | Major | suspicious

`#pragma pack` is a non standard extension used to specify the packing alignment for structure, union and class members.

It is useful to

- remove padding and decrease the size of objects
- align members to better fit optimal cpu alignment

However, the pragma pack directives need to be correctly defined to work properly.

This rule raises an issue if:

- the specified packing value is incorrect: it can only be 1, 2, 4, 8, or 16
- a parameter is ill-formed
- the `pop` variant of this `#pragma` is called with both arguments `identifier` and `value`: such a call is undefined behavior
- a `#pragma pack(push...)` is performed but there is not corresponding use of `#pragma pack(pop...)`
- a `#pragma pack(pop...)` is performed but there is not corresponding use of `#pragma pack(push...)`
- a `#pragma pack` is in effect across several files: this becomes too complex, and could easily lead to undefined behavior, the same structure having different layout when seen from different translation units

Noncompliant Code Example

```
#pragma pack(5) // Noncompliant, value is invalid
#pragma pack(2+2) // Noncompliant, value should be a literal

#pragma pack(4)
#include "myFile.h" // Noncompliant, the specified alignement

struct T {
   int i;
   short j;
   double k;
};

#pragma pack(push, r1, 16)
#pragma pack(pop, r1, 4)  // Noncompliant, call to pop with t

#pragma pack(push, r2, 16)
#pragma pack(pop, r3)  // Noncompliant, call to pop with no m

#pragma pack(push, 8) // Noncompliant, unmatched push
```

Compliant Solution

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

```
#include "myFile.h"

#pragma pack(4)

struct T {
   int i;
   short j;
   double k;
};

#pragma pack(push, r1, 16)
#pragma pack(pop, r1)

#pragma pack(push, r2, 16)
#pragma pack(pop, r2)

#pragma pack(push, 8)
#pragma pack(pop)
```

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition