sonar RULES

Products

- 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

Size of bit fields should not exceed the size of their types

Analyze your code

Code Smell | Major | bad-practice

Bit fields allow the developers to declare a class member with a specific size.

However, the size of a bit field is also constrained by its type: even if the specified size is greater than the size of the type, the value of the bit field will not exceed the maximum value of this type. The extra bits will just create unused padding.

The incompatibility of the size of the type with the specified size can have two causes: either the specified size is a typo error (that is the most probable cause) or the developer did not realize the size of the type he chose was too small.

Noncompliant Code Example

```
class A {
  unsigned int b : 55; // Noncompliant, specified size is gre
};
```

Compliant Solution

```
class A {
  unsigned int b : 32;
};
```

Or

```
class A {
  unsigned long long int b : 55;
};
```

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