

- 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

"memcpy" should only be called with pointers to trivially copyable types with no padding

Bug

Dynamically allocated memory should be released

Analyze your code

Bug Blocker

cwe symbolic-execution leak denial-of-service cert

Memory allocated dynamically with `calloc(...)`, `malloc(...)`, `realloc(...)` or `new` should be released when it's not needed anymore. Failure to do so will result in a memory leak that could bring the box to its knees.

This rule raises an issue when memory is allocated and not freed in the same function. Allocated memory is ignored if a pointer to it is returned to the caller or stored in a structure that's external to the function.

Noncompliant Code Example

```
int fun() {
    char* name = (char *) malloc (size);
    if (!name) {
        return 1;
    }
    // ...
    return 0; // Noncompliant, memory pointed by "name" has not
}
```

Compliant Solution

```
int fun() {
    char* name = (char *) malloc (size);
    if (!name) {
        return 1;
    }
    // ...
    free(name);
    return 0;
}
```

See

- [MITRE, CWE-401](#) - Improper Release of Memory Before Removing Last Reference ('Memory Leak')
- [MEM00-C](#) - Allocate and free memory in the same module, at the same level of abstraction
- [CERT, MEM31-C](#) - Free dynamically allocated memory when no longer needed

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