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

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

Null pointers should not be dereferenced

Analyze your code

Bug Major cwe symbolic-execution cert

A pointer to null (the 0 memory address) should never be dereferenced/accessed. Doing so will at best cause abrupt program termination, without the ability to run any cleanup processes. At worst, it could expose debugging information that would be useful to an attacker or it could allow an attacker to bypass security measures.

Noncompliant Code Example

```
char *p1 = ... ;
if (p1 == NULL && *p1 == '\t') { // Noncompliant, p1 will be
  // ...
}

char *p2 = ... ;
if (p2 != NULL) {
    // ...
}
*p2 = '\t'; // Noncompliant; potential null-dereference

char *p3, *p4;
p3 = NULL;
// ...
p4 = p3;
*p4 = 'a';  // Noncompliant
```

Compliant Solution

```
char *p1 = ... ;
if (p1 != NULL && *p1 == '\t') { // Compliant, *p1 cannot be
  // ...
}

char *p2 = ... ;
if (p2 != NULL) {
    // ...
  *p2 = '\t'; // Compliant
}
```

See

- MITRE, CWE-476 - NULL Pointer Dereference
- CERT, EXP34-C. - Do not dereference null pointers
- CERT, EXP01-J. - Do not use a null in a case where an object is required

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition