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 578 | Vulnerability 13 | Bug 111 | Security Hotspot 18 | Code Smell 436 | Quick Fix 68

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

Assigning to an optional should directly target the optional
Bug

Result of the standard remove algorithms should not be ignored
Bug

"std::scoped_lock" should be created with constructor arguments
Bug

Objects should not be sliced
Bug

Immediately dangling references should not be created
Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked
Bug

"pthread_mutex_t" should be properly

Memory access should be explicitly bounded to prevent buffer overflows

Analyze your code

Bug | Blocker | cwe symbolic-execution cert

Array overruns and buffer overflows happen when memory access accidentally goes beyond the boundary of the allocated array or buffer. These overreaching accesses cause some of the most damaging, and hard to track defects.

Noncompliant Code Example

```
int array[10];
array[10] = 0; // Noncompliant: index should be between 0 & 9

char *buffer1 = (char *) malloc(100);
char *buffer2 = (char *) malloc(50);
memcpy(buffer2, buffer1, 100); // Noncompliant: buffer2 will
```

Compliant Solution

```
int array[10];
array[9] = 0;

char *buffer1 = (char *) malloc(100);
char *buffer2 = (char *) malloc(50);
memcpy(buffer2, buffer1, 50);
```

See

- MITRE, CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer
- MITRE, CWE-131 - Incorrect Calculation of Buffer Size
- MITRE, CWE-788 - Access of Memory Location After End of Buffer
- CERT, ARR30-C. - Do not form or use out-of-bounds pointers or array subscripts
- CERT, STR50-CPP. - Guarantee that storage for strings has sufficient space for character data and the null terminator

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

initialized and destroyed

Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Bug

"std::move" and "std::forward" should not be confused

Bug

A call to "wait()" on a "std::condition_variable" should have a