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 initialized and destroyed
Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Octal values should not be used

Analyze your code

Code Smell | Blocker | Quick Fix | based-on-misra cert pitfall

Integer literals starting with a zero are octal rather than decimal values. While using octal values is fully supported, most developers do not have experience with them. They may not recognize octal values as such, mistaking them instead for decimal values.

Hexadecimal literals (`0xdeadbeef`) and binary literals (`0b0101'0110'00011`, available since C++14), on the other hand, have a clear marker (`0x` or `0b`) and can be used to define the binary representation of a value.

Character literals starting with `\` and followed by one to three digits are octal escaped literals. Character literals starting with `\x` and followed by one or more hexits are hexadecimal escaped literals, and are usually more readable.

Noncompliant Code Example

```
int myNumber = 010;   // Noncompliant. myNumber will hold 8,

char myChar = '\40'; // Noncompliant. myChar will hold 32 rat
```

Compliant Solution

```
int myNumber = 8; // Use decimal when representing the value
// or
int myNumber = 0b1000; // Use binary or hexadecimal for a bit

char myChar = '\x20'; // Use hexadecimal
// or
char myChar = '\n'; // Use the common notation if it exists f
```

Exceptions

- Octal values have traditionally been used for user permissions in Posix file systems, and this rule will ignore octal literals used in this context.
- `'\0'` is a common notation for a null character, so the rule ignores it.

See

- MISRA C:2004, 7.1 - Octal constants (other than zero) and octal escape sequences shall not be used.
- MISRA C++:2008, 2-13-2 - Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used
- MISRA C:2012, 7.1 - Octal constants shall not be used
- CERT, DCL18-C. - Do not begin integer constants with 0 when specifying a decimal value
- CERT, DCL50-J. - Use visually distinct identifiers

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

Bug

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

Bug

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

Bug

A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast

Bug

Functions with "noreturn" attribute should not return

Bug

RAII objects should not be temporary

Bug

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

Bug

"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types

Bug

"std::auto_ptr" should not be used

Bug

Destructors should be "noexcept"

Bug