

- 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

### Function exit paths should have appropriate return values

Analyze your code

Bug Critical cwe based-on-misra cert

Every call to a function with a non-void return type is expected to return some value. Including a return path in a non-void function that does not explicitly return a value results in undefined behavior.

Conversely, every call to a function with a void return type is expected to not return any value. Returning a value from a void function probably indicates a programming error.

#### Noncompliant Code Example

```
int my_func (int a)
{
    if (a > 100)
    {
        return; // Noncompliant
    }

    if (a > 80)
    {
        throw new Exception(); // Compliant
    }

    // Noncompliant
}
```

#### Compliant Solution

```
int my_func (int a)
{
    if (a > 100)
    {
        return 12;
    }

    if (a > 80)
    {
        throw new Exception();
    }

    return a;
}
```

#### Exceptions

This rule doesn't raise an exception when the return statement for a void function, is itself a void expression.

```
void foo() {
    // Do stuff ...
}
```

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

```
}

void bar() {
    return foo();
}
```

Furthermore, the issue is not raised for the coroutines, introduced in `{cpp20}`, that always declare coroutine object as a return type, but returned object is implicitly created by compiler. The coroutine body itself may never contains `return` statement (the use of it is disallowed), and `co_return` is used for coroutine that returns a value (define `return_value` in promise-type).

**See**

- `{rule:cpp:S6369}` - Coroutine should have `co_return` on each execution path or provide `return_void`
- MISRA C:2004, 16.8 - All exit paths from a function with non-void return type shall have an explicit return statement with an expression
- MISRA C++:2008, 8-4-3 - All exit paths from a function with non-void return type shall have an explicit return statement with an expression
- MISRA C:2012, 17.4 - All exit paths from a function with non-void return type shall have an explicit return statement with an expression
- [MITRE, CWE-394](#) - Unexpected Status Code or Return Value
- [CERT, MSC37-C](#) - Ensure that control never reaches the end of a non-void function
- [CERT, MSC52-CPP](#) - Value-returning functions must return a value from all exit paths
- [CERT, MSC53-CPP](#) - Do not return from a function declared `[[noreturn]]`

Available In:

 |  |  Developer Edition