



Apex

**ABAP** 

С



CloudFormation

COBOL

C#

**CSS** 

Flex

=GO

5 HTML

Go

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

ΑII 578 6 Vulnerability 13 rules

**R** Bug (111)

• Security Hotspot

⊗ Code (436)

Quick 68 Fix

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

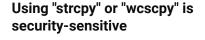
"pthread\_mutex\_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread\_mutex\_t" should be properly initialized and destroyed

📆 Bug

"pthread\_mutex\_t" should not be consecutively locked or unlocked twice



Analyze your code

cwe owasp cert

In C, a string is just a buffer of characters, normally using the null character as a sentinel for the end of the string. This means that the developer has to be aware of low-level details such as buffer sizes or having an extra character to store the final null character. Doing that correctly and consistently is notoriously difficult and any error can lead to a security vulnerability, for instance, giving access to sensitive data or allowing arbitrary code execution.

The function char \*strcpy(char \* restrict dest, const char \* restrict src); copies characters from src to dest. The wcscpy does the same for wide characters and should be used with the same guidelines.

Note: the functions strncpy and wcsncpy might look like attractive safe replacements for strcpy and wcscpy, but they have their own set of issues (see {rule:cpp:S5816}), and you should probably prefer another more adapted alternative.

#### **Ask Yourself Whether**

- There is a possibility that either the source or the destination pointer is null
- There is a possibility that the source string is not correctly null-terminated, or that its length (including the final null character) can be larger than the size of the destination buffer.
- There is an overlap between source and destination

There is a risk if you answered yes to any of those questions.

#### **Recommended Secure Coding Practices**

- C11 provides, in its annex K, the strcpy\_s and the wcscpy\_s that were designed as safer alternatives to strcpy and wcscpy. It's not recommended to use them in all circumstances, because they introduce a runtime overhead and require to write more code for error handling, but they perform checks that will limit the consequences of calling the function with bad arguments.
- Even if your compiler does not exactly support annex K, you probably have access to similar functions, for example, strlcpy in FreeBSD
- If you are writing C++ code, using std::string to manipulate strings is much simpler and less error-prone

## **Sensitive Code Example**

```
int f(char *src) {
char dest[256];
strcpy(dest, src); // Sensitive: might overflow
return doSomethingWith(dest);
```

### **Compliant Solution**

```
int f(char *src) {
char *dest = malloc(strlen(src) + 1); // For the final 0
strcpy(dest, src); // Compliant: we made sure the buffer is
int r= doSomethingWith(dest);
free(dest);
return r;
```

See



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

Rug 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

Rug Bug

"std::auto\_ptr" should not be used

📆 Bug

Destructors should be "noexcept"

📆 Bug

- OWASP Top 10 2021 Category A6 Vulnerable and Outdated Components
- OWASP Top 10 2017 Category A9 Using Components with Known Vulnerabilities
- MITRE, CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- CERT, STR07-C. Use the bounds-checking interfaces for string manipulation

Available In:

sonarcloud 🚳 | sonarqube | Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of  $Sonar Source \ S.A. \ All \ other \ trademarks \ and \ copyrights \ are \ the \ property \ of \ their \ respective$ owners. All rights are expressly reserved. **Privacy Policy**