

Objective C

PHP PL/I

PL/SQL

Python

**RPG** 

Ruby

Scala

Swift

Terraform

日 Text

TypeScript

T-SQL

**VB.NET** 

VB6

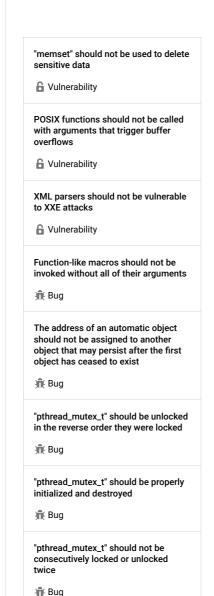
XML



# C static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C code

⊗ Code (206) O Quick 14 ΑII 311 Security 18 6 Vulnerability (13) ₩ Bug (74) rules Hotspot Smell



Functions with "noreturn" attribute

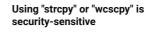
"memcmp" should only be called with pointers to trivially copyable types

should not return

with no padding

₩ Bua

🖷 Bug



Tags

Analyze your code

cwe owasp cert

Search by name.

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 stropy and woschy. 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
```

Stack allocated memory and nonowned 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

```
int r= doSomethingWith(dest);
 free(dest);
 return r;
}
```

#### See

- 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 SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy