rules.sonarsource.com

C++ static code analysis: Using "strncat" or "wcsncat" is security-sensitive

3-4 minutes

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 *strncat( char *restrict dest, const char *restrict src, size_t count );` appends the characters of string `src` at the end of `dest`, but only add `count` characters max. `dest` will always be `null`-terminated. The `wcsncat` does the same for wide characters, and should be used with the same guidelines.

Ask Yourself Whether

- There is a possibility that either the `src` or the `dest` pointer is `null`
- The current string length of `dest` plus the current string length of `src` plus `1` (for the final `null` character) is larger than the size of the buffer pointer-to by `src`
- There is a possibility that either string is not correctly `null`-terminated

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

Recommended Secure Coding Practices

- C11 provides, in its annex K, the `strncat_s` and the `wcsncat_s` that were designed as safer alternatives to `strncat` and `wcsncat`. 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
- If you are using `strncat` and `wsncat` as a safer version of `strcat` and `wcscat`, you should instead consider `strcat_s` and `wcscat_s` because these functions have several shortcomings:
  - It's not easy to detect truncation
  - The `count` parameter is error-prone
  - Computing the `count` parameter typically requires computing the string length of `dest`, at which point other simpler alternatives exist

Sensitive Code Example

```
int f(char *src) {
  char dest[256];
  strcpy(dest, "Result: ");
  strncat(dest, src, sizeof dest); // Sensitive: passing the buffer size instead of the remaining size
  return doSomethingWith(dest);
}
```

Compliant Solution

```
int f(char *src) {
  char result[] = "Result: ";
  char dest[256];
  strcpy(dest, result);
  strncat(dest, src, sizeof dest - sizeof result); // Compliant but may silently truncate
  return doSomethingWith(dest);
}
```

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: