rules.sonarsource.com

C static code analysis: "sprintf" should not be used

3 minutes

When using `sprintf` , it’s up to the developer to make sure the size of the buffer to be written to is large enough to avoid buffer overflows. Buffer overflows can cause the program to crash at a minimum. At worst, a carefully crafted overflow can cause malicious code to be executed.

Ask Yourself Whether

- if the provided buffer is large enough for the result of any possible call to the `sprintf` function (including all possible format strings and all possible additional arguments).

There is a risk if you answered no to the above question.

Recommended Secure Coding Practices

There are fundamentally safer alternatives. `snprintf` is one of them. It takes the size of the buffer as an additional argument, preventing the function from overflowing the buffer.

- Use `snprintf` instead of `sprintf`. The slight performance overhead can be afforded in a vast majority of projects.
- Check the buffer size passed to `snprintf`.

If you are working in C++, other safe alternative exist:

- `std::string` should be the prefered type to store strings
- You can format to a string using `std::ostringstream`
- Since C++20, `std::format` is also available to format strings

Sensitive Code Example

sprintf(str, "%s", message);   // Sensitive: `str` buffer size is not checked and it is vulnerable to overflows

Compliant Solution

snprintf(str, sizeof(str), "%s", message); // Prevent overflows by enforcing a maximum size for `str` buffer

Exceptions

It is a very common and acceptable pattern to compute the required size of the buffer with a call to `snprintf` with the same arguments into an empty buffer (this will fail, but return the necessary size), then to call `sprintf` as the bound check is not needed anymore. Note that 1 needs to be added by the size reported by `snprintf` to account for the terminal null character.

```
size_t buflen = snprintf(0, 0, "%s", message);
char* buf = malloc(buflen + 1); // For the final 0
sprintf(buf, "%s", message);{code}
```

See

- OWASP Top 10 2021 Category A6 - Vulnerable and Outdated Components
- OWASP Top 10 2017 Category A9 - Using Components with Known Vulnerabilities
- MITRE, CWE-676 - Use of Potentially Dangerous Function
- MITRE, CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer
- SANS Top 25 - Risky Resource Management

Available In: