API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size

Functions that have an array as a parameter should also have an additional parameter that indicates the maximum number of elements that can be stored in the array. That parameter is required to ensure that the function does not access memory outside the bounds of the array and adversely influence program execution. It should be present for each array parameter (in other words, the existence of each array parameter implies the existence of a complementary parameter that represents the maximum number of elements in the array).

Note that array is used in this recommendation to mean array, string, or any other pointer to a contiguous block of memory in which one or more elements of a particular type are (potentially) stored. These terms are all effectively synonymous and represent the same potential for error.

Also note that this recommendation suggests the parameter accompanying array parameters indicates the maximum number of elements that can be stored in the array, not the maximum size, in bytes, of the array, because

1. It does not make sense to think of array sizes in bytes in all cases—for example, in the case of an array of integers.
2. If the size in bytes of the array is required, it can be derived from the number of elements in the array.
3. It is better not to add to the cognitive load of the function user by requiring the user to calculate the size in bytes of the array.

In most cases, the distinction between the number of elements and number of bytes is moot: there is a clear mapping between the two, and it is easier to think in terms of number of elements anyway. Unfortunately, this issue can become muddled when working with multibyte strings because the logical entity being manipulated differs from that of the type being used to implement it. Here, it is important to remember that the type of the array is a character, not a multibyte character. Accordingly, the number of elements in the array is represented as a number of characters.

Noncompliant Code Example

It is not necessary to go beyond the standard C library to find examples that violate this recommendation because the C language often prioritizes performance at the expense of robustness. The following are two examples from the C Standard, subclause 7.24 [ISO/IEC 9899:2011]:

```c
char *strncpy(char * restrict s1, const char * restrict s2, size_t n);
char *strncat(char * restrict s1, const char * restrict s2, size_t n);
```

These functions have two problems. First, there is no indication of the size of the first array, `s1`. As a result, it is not possible to discern within the function how large `s1` is and how many elements may be written into it. Second, it appears that a size is supplied for `s2` but the `size_t` parameter `n` actually gives the number of elements to copy. Consequently, there is no way for either function to determine the size of the array `s2`.

Compliant Solution

The C `strncpy()` and `strncat()` functions could be improved by adding element count parameters as follows:

```c
char *improved_strncpy(char * restrict s1, size_t s1count, const char * restrict s2, size_t s2count, size_t n);
char *improved_strncat(char * restrict s1, size_t s1count, const char * restrict s2, size_t s2count, size_t n);
```

The `n` parameter is used to specify a number of elements to copy that is less than the total number of elements in the source string.

Compliant Solution (C11 Annex K)

The C Standard, Annex K (normative) Bounds-checking interfaces, defines bounds-checking versions of standard C library string-handling functions:

```c
errno_t strncpy_s(char * restrict s1, rsize_t s1max, const char * restrict s2, rsize_t n);
errno_t strcat_s(char * restrict s1, rsize_t s1max, const char * restrict s2);
```

There are two notable differences between the compliant solution and the secure versions from Annex K. First, the Annex K versions use `rsize_t` instead of `size_t`, which allows the size to be compared against the reasonable limit for a single object, `RSIZE_MAX`. Second, the Annex K versions do not require an element count for the second array. Consequently, these functions have limited ability to validate the input for `s2`. However, a size value for `s1` is required, so memory outside of the range for `s1` should not be overwritten.

Exceptions

API02-C-EX1: Functions that can guarantee via their runtime-constraint handlers that no out-of-bounds read or write occurs may omit the maximum-elements argument. For instance, the `s2` parameter to `strcat_s()` needs no `max` parameter.

```c
errno_t strcat_s(char * restrict s1, rsize_t s1max, const char * restrict s2);
```
As another example, consider `strcpy_s()`:

```
errno_t strcpy_s(char * restrict s1, rsize_t s1max, const char * restrict s2);
```

This function provides no explicit maximum argument to `s2`. However, it requires that `s1max` be larger than `s2`, thereby preventing an out-of-bounds read.

**Risk Assessment**

Failure to follow this recommendation can result in improper memory accesses and buffer overflows that are detrimental to the correct and continued execution of the program.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>API02-C</td>
<td>High</td>
<td>Likely</td>
<td>High</td>
<td>P9</td>
<td>L2</td>
</tr>
</tbody>
</table>

**Automated Detection**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSonar</td>
<td>8.0p0</td>
<td>BADFUNC.BO.*</td>
<td>A collection of checks that report uses of library functions prone to internal buffer overflows.</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>2023.1</td>
<td>CERT_C-API02-a CERT_C-API02-b</td>
<td>Avoid using unsafe string functions which may cause buffer overflows. Don't use unsafe C functions that do write to range-unchecked buffers.</td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

**Related Guidelines**

*Key here* (explains table format and definitions)

**Bibliography**