Proposal for C23
WG14 N 2810

Title:	calloc overflow handling
Author, affiliation:	Robert C. Seacord, NCC Group
Date:	2021-10-4
Proposal category:	Defect
Target audience:	Implementers
Abstract:	Explicitly define behavior on overflow in calloc
Prior art:	C

calloc overflow handling

Reply-to: Robert C. Seacord (rcseacord@gmail.com)
Document No: **N 2810**
Reference Document: N 2800
Date: 2021-10-4

The reference document N 2800 was discussed at the August/September 2021 meeting [N2803]. A straw poll was conducted if the committee would support something along the lines of the second wording in N2800? The result was 18-0-3; a clear sentiment to go in that direction. This paper pursues this direction, addressing the identified problems.

Change Log

2021-8-30:

- Initial version (N 2800)

2021-9-11:

- Eliminated first wording alternative
- Insert “mathematical product”

Introduction and Rationale

The current wording in the working draft (N2596 7.22.3.2.2) describes `calloc` as follows: "The `calloc` function allocates space for an array of `nmemb` objects, each of whose size is `size`. The space is initialized to all bits zero.” In particular, regardless of whether or not the C expression `nmemb * size` overflows, `calloc` is not permitted to return a non-null pointer to fewer bytes than the mathematical product of `nmemb` and `size` (i.e., assuming infinitely ranged integers). According to Subclause 7.22.3.2 p2 “The `calloc` function allocates space for an array of `nmemb` objects, each of whose size is `size`” and according to Subclause 7.22.3 p1 “If the space cannot be allocated, a null pointer is returned.” Implementations where overflow returned non-null values are non-conforming in the current draft standard.

Security
RUS-CERT [RUS-CERT 2002, Weimer 2002] documented the defect in calloc implementations and similar routines:

> *Integer overflow can occur during the computation of the memory region size by calloc and similar functions. As a result, the function returns a buffer which is too small, possibly resulting in a subsequent buffer overflow.*

While most implementations were repaired, the standard was not updated to clarify the existing requirement.

The problem subsequently reoccurred [MSRC 2021]. The same vulnerability exists in standard memory allocation functions spanning widely used real-time operating systems (RTOS), embedded software development kits (SDKs), and C standard library (libc) implementations. These findings have been shared with vendors through responsible disclosure led by the Microsoft Security Response Center (MSRC) and the Department of Homeland Security (DHS), enabling these vendors to investigate and patch the vulnerabilities.

For a full list of affected products and CVEs, please visit the DHS website: ICSA-21-119-04 Multiple RTOS (https://us-cert.cisa.gov/ics/advisories/icsa-21-119-04).

The primary purpose of this proposal is to clarify the existing behavior of `calloc` in the event that `nmemb * size` overflows, to help prevent future implementation defects resulting in security vulnerabilities.

Huge Objects

This paper makes it explicit that the `calloc` function cannot be used to allocate huge objects (objects for which the size cannot be represented as a `size_t`). No such implementations have been identified.

Four families of implementations that are being actively developed have been identified and are conforming that have pointers wider than `size_t`:

- AS/400 and follow-ups
- some versions of Elbrus
- SDC for some targets
- gcc for the M32C target

For all of them the reason seems to be that they are storing additional information in the pointers, such as segments or even type information, not that the processor would be able to address objects that are larger than `SIZE_MAX`.

The first two have 128 bit pointers and don't seem to have `uintptr_t` defined. Maybe this is an indirect consequence of our problematic definition of `intmax_t` which might be hindering those platforms to provide 128 bit integer types.

The Small Device C Compiler (SDCC) supports some targets where `uintptr_t` is wider than `size_t`. For example, the MCS-51 (a microcontrollers architecture) there are 3 disjoint intrinsic named address spaces: `__idata`, `__xdata` and `__code` (there are more intrinsic named address spaces, but each is a subset of one of these three). The first has 8-bit addresses, the latter each have 16-bit addresses. The 16-bit `size_t` is sufficient for the size of any object. The larger `uintptr_t` uses the upper bits to decide which of the three spaces the pointer points into. Both `void *` and `uintptr_t` are wider than `size_t`. So although `uintptr_t` is wider than `size_t` on this implementation, its width is unrelated to the size of objects that can be allocated and consequently does not prevent the definition of calloc to wraparound.

`__builtin_object_size` as implemented by gcc and llvm assumes `SIZE_MAX` as a failure, which implies that actual allocations in practice are always `< SIZE_MAX`. If such an implementation did exist, the allocation and management of such objects is outside the scope of the C Standard library functions. For example, the `sizeof` operator returns a result whose type is `size_t` (7.19, p2). As such, it would not be possible to use the `sizeof` operator to determine the size of such a huge object.

There are other platforms with unconventional sizes, but they are not actively being developed.

Proposed Wording

The wording proposed is a diff from WG14 N2596. Green text is new text, while ~~red text~~ is deleted text.

The **calloc** function returns either ~~a null pointer or~~ a pointer to the allocated space or a null pointer if the space cannot be allocated or if the mathematical product of `nmemb * size` is not representable as a value of type `size_t`.

4.0 Acknowledgements

I would like to recognize the following people for their help with this work: Jens Gustedt, David Goldblatt, Mark Santaniello, Florian Weimer, Erik Steringer, Philipp Klaus Krause, and Aaron Ballman.

5.0 References

[RUS-CERT 2002] RUS-CERT Advisory. Flaw in calloc and similar routines 2002-08:02. URL: https://web.archive.org/web/20081225201357/http://cert.uni-stuttgart.de/advisories/calloc.php

[Weimer 2002] Florian Weimer. RUS-CERT Advisory 2002-08:02: Flaw in calloc and similar routines. Mon, 05 Aug 2002. URL: https://www.opennet.ru/base/cert/1028651886_905.txt.html

[MSRC 2021] MSRC Team. "BadAlloc" – Memory allocation vulnerabilities could affect wide range of IoT and OT devices in industrial, medical, and enterprise networks. April 29, 2021. URL: https://msrc-blog.microsoft.com/2021/04/29/badalloc-memory-allocation-vulnerabilities-could-affect-wide-range-of-iot-and-ot-devices-in-industrial-medical-and-enterprise-networks/

[N2803] Draft Minutes for 27 and 30 – 31 August, 1 – 3 September, 2021. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2803.pdf