# CWE Detail – CWE-763

## Description

The product attempts to return a memory resource to the system, but it calls the wrong release function or calls the appropriate release function incorrectly.

## Extended Description

This weakness can take several forms, such as: The memory was allocated, explicitly or implicitly, via one memory management method and deallocated using a different, non-compatible function (CWE-762). The function calls or memory management routines chosen are appropriate, however they are used incorrectly, such as in CWE-761.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2019-11930: function "internally calls 'calloc' and returns a pointer at an index... inside the allocated buffer. This led to freeing invalid memory."

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Modify Memory, DoS: Crash, Exit, or Restart, Execute Unauthorized Code or Commands — Notes: This weakness may result in the corruption of memory, and perhaps instructions, possibly leading to a crash. If the corrupted memory can be effectively controlled, it may be possible to execute arbitrary code.

## Potential Mitigations

**•** Implementation: Only call matching memory management functions. Do not mix and match routines. For example, when you allocate a buffer with malloc(), dispose of the original pointer with free(). (Effectiveness: N/A)

**•** Implementation: When programming in C++, consider using smart pointers provided by the boost library to help correctly and consistently manage memory. (Effectiveness: N/A)

**•** Architecture and Design: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, glibc in Linux provides protection against free of invalid pointers. (Effectiveness: N/A)

**•** Architecture and Design: Use a language that provides abstractions for memory allocation and deallocation. (Effectiveness: N/A)

**•** Testing: Use a tool that dynamically detects memory management problems, such as valgrind. (Effectiveness: N/A)

## Applicable Platforms

**•** C (Class: None, Prevalence: Often)

**•** C++ (Class: None, Prevalence: Often)

## Demonstrative Examples

**•** Since strsep is not allocating any new memory, freeing an element in the middle of the array is equivalent to free a pointer in the middle of inputstring.

**•** Instead, the programmer should have either created the object with one of the malloc family functions, or else deleted the object with the delete operator.

**•** However, if the character is not at the beginning of the string, or if it is not in the string at all, then the pointer will not be at the start of the buffer when the programmer frees it.

**•** While the above code attempts to free memory associated with bad commands, since the memory was all allocated in one chunk, it must all be freed together.

## Notes

**•** Maintenance: The view-1000 subtree that is associated with this weakness needs additional work. Several entries will likely be created in this branch. Currently the focus is on free() of memory, but delete and other related release routines may require the creation of intermediate entries that are not specific to a particular function. In addition, the role of other types of invalid pointers, such as an expired pointer, i.e. CWE-415 Double Free and release of uninitialized pointers, related to CWE-457.