# CWE Detail – CWE-789

## Description

The product allocates memory based on an untrusted, large size value, but it does not ensure that the size is within expected limits, allowing arbitrary amounts of memory to be allocated.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2022-21668: Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190).

**•** CVE-2010-3701: program uses ::alloca() for encoding messages, but large messages trigger segfault

**•** CVE-2008-1708: memory consumption and daemon exit by specifying a large value in a length field

**•** CVE-2008-0977: large value in a length field leads to memory consumption and crash when no more memory is available

**•** CVE-2006-3791: large key size in game program triggers crash when a resizing function cannot allocate enough memory

**•** CVE-2004-2589: large Content-Length HTTP header value triggers application crash in instant messaging application due to failure in memory allocation

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Resource Consumption (Memory) — Notes: Not controlling memory allocation can result in a request for too much system memory, possibly leading to a crash of the application due to out-of-memory conditions, or the consumption of a large amount of memory on the system.

## Potential Mitigations

**•** Implementation: Perform adequate input validation against any value that influences the amount of memory that is allocated. Define an appropriate strategy for handling requests that exceed the limit, and consider supporting a configuration option so that the administrator can extend the amount of memory to be used if necessary. (Effectiveness: N/A)

**•** Operation: Run your program using system-provided resource limits for memory. This might still cause the program to crash or exit, but the impact to the rest of the system will be minimized. (Effectiveness: N/A)

## Applicable Platforms

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

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

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** Suppose an attacker provides a size value of:

**•** The HashMap constructor will verify that the initial capacity is not negative, however there is no check in place to verify that sufficient memory is present. If the attacker provides a large enough value, the application will run into an OutOfMemoryError.

**•** Since a and b are declared as signed ints, the "a - b" subtraction gives a negative result (-1). However, since len is declared to be unsigned, len is cast to an extremely large positive number (on 32-bit systems - 4294967295). As a result, the buffer buf[len] declaration uses an extremely large size to allocate on the stack, very likely more than the entire computer's memory space.

**•** The buffer length ends up being -1, resulting in a blown out stack. The space character after the colon is included in the function calculation, but not in the caller's calculation. This, unfortunately, is not usually so obvious but exists in an obtuse series of calculations.

**•** The index is not validated at all (CWE-129), so it might be possible for an attacker to modify an element in @messages that was not intended. If an index is used that is larger than the current size of the array, the Perl interpreter automatically expands the array so that the large index works.

**•** N/A

## Notes

**•** Relationship: This weakness can be closely associated with integer overflows (CWE-190). Integer overflow attacks would concentrate on providing an extremely large number that triggers an overflow that causes less memory to be allocated than expected. By providing a large value that does not trigger an integer overflow, the attacker could still cause excessive amounts of memory to be allocated.

**•** Applicable Platform: Uncontrolled memory allocation is possible in many languages, such as dynamic array allocation in perl or initial size parameters in Collections in Java. However, languages like C and C++ where programmers have the power to more directly control memory management will be more susceptible.