# CWE Detail – CWE-839

## Description

The product checks a value to ensure that it is less than or equal to a maximum, but it does not also verify that the value is greater than or equal to the minimum.

## Extended Description

Some products use signed integers or floats even when their values are only expected to be positive or 0. An input validation check might assume that the value is positive, and only check for the maximum value. If the value is negative, but the code assumes that the value is positive, this can produce an error. The error may have security consequences if the negative value is used for memory allocation, array access, buffer access, etc. Ultimately, the error could lead to a buffer overflow or other type of memory corruption. The use of a negative number in a positive-only context could have security implications for other types of resources. For example, a shopping cart might check that the user is not requesting more than 10 items, but a request for -3 items could cause the application to calculate a negative price and credit the attacker's account.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2010-1866: Chain: integer overflow (CWE-190) causes a negative signed value, which later bypasses a maximum-only check (CWE-839), leading to heap-based buffer overflow (CWE-122).

**•** CVE-2009-1099: Chain: 16-bit counter can be interpreted as a negative value, compared to a 32-bit maximum value, leading to buffer under-write.

**•** CVE-2011-0521: Chain: kernel's lack of a check for a negative value leads to memory corruption.

**•** CVE-2010-3704: Chain: parser uses atoi() but does not check for a negative value, which can happen on some platforms, leading to buffer under-write.

**•** CVE-2010-2530: Chain: Negative value stored in an int bypasses a size check and causes allocation of large amounts of memory.

**•** CVE-2009-3080: Chain: negative offset value to IOCTL bypasses check for maximum index, then used as an array index for buffer under-read.

**•** CVE-2008-6393: chain: file transfer client performs signed comparison, leading to integer overflow and heap-based buffer overflow.

**•** CVE-2008-4558: chain: negative ID in media player bypasses check for maximum index, then used as an array index for buffer under-read.

## Common Consequences

**•** Impact: Modify Application Data, Execute Unauthorized Code or Commands — Notes: An attacker could modify the structure of the message or data being sent to the downstream component, possibly injecting commands.

**•** Impact: DoS: Resource Consumption (Other) — Notes: in some contexts, a negative value could lead to resource consumption.

**•** Impact: Modify Memory, Read Memory — Notes: If a negative value is used to access memory, buffers, or other indexable structures, it could access memory outside the bounds of the buffer.

## Potential Mitigations

**•** Implementation: If the number to be used is always expected to be positive, change the variable type from signed to unsigned or size\_t. (Effectiveness: N/A)

**•** Implementation: If the number to be used could have a negative value based on the specification (thus requiring a signed value), but the number should only be positive to preserve code correctness, then include a check to ensure that the value is positive. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** The code performs a check to make sure that the packet does not contain too many headers. However, numHeaders is defined as a signed int, so it could be negative. If the incoming packet specifies a value such as -3, then the malloc calculation will generate a negative number (say, -300 if each header can be a maximum of 100 bytes). When this result is provided to malloc(), it is first converted to a size\_t type. This conversion then produces a large value such as 4294966996, which may cause malloc() to fail or to allocate an extremely large amount of memory (CWE-195). With the appropriate negative numbers, an attacker could trick malloc() into using a very small positive number, which then allocates a buffer that is much smaller than expected, potentially leading to a buffer overflow.

**•** This code first exhibits an example of CWE-839, allowing "s" to be a negative number. When the negative short "s" is converted to an unsigned integer, it becomes an extremely large positive integer. When this converted integer is used by strncpy() it will lead to a buffer overflow (CWE-119).

**•** However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.

**•** The withdraw method includes a check to ensure that the withdrawal amount does not exceed the maximum limit allowed, however the method does not check to ensure that the withdrawal amount is greater than a minimum value (CWE-129). Performing a range check on a value that does not include a minimum check can have significant security implications, in this case not including a minimum range check can allow a negative value to be used which would cause the financial application using this class to deposit money into the user account rather than withdrawing. In this example the if statement should the modified to include a minimum range check, as shown below.