# CWE Detail – CWE-190

## Description

The product performs a calculation that can
 produce an integer overflow or wraparound when the logic
 assumes that the resulting value will always be larger than
 the original value. This occurs when an integer value is
 incremented to a value that is too large to store in the
 associated representation. When this occurs, the value may
 become a very small or negative number.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2025-27363: Font rendering library does not properly
 handle assigning a signed short value to an unsigned
 long (CWE-195), leading to an integer wraparound
 (CWE-190), causing too small of a buffer (CWE-131),
 leading to an out-of-bounds write
 (CWE-787). (KEV)

**•** CVE-2021-43537: Chain: in a web browser, an unsigned 64-bit integer is forcibly cast to a 32-bit integer (CWE-681) and potentially leading to an integer overflow (CWE-190). If an integer overflow occurs, this can cause heap memory corruption (CWE-122)

**•** 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-2022-0545: Chain: 3D renderer has an integer overflow (CWE-190) leading to write-what-where condition (CWE-123) using a crafted image.

**•** CVE-2021-30860: Chain: improper input validation (CWE-20) leads to integer overflow (CWE-190) in mobile OS, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2021-30663: Chain: improper input validation (CWE-20) leads to integer overflow (CWE-190) in mobile OS, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2018-10887: Chain: unexpected sign extension (CWE-194) leads to integer overflow (CWE-190), causing an out-of-bounds read (CWE-125)

**•** CVE-2019-1010006: Chain: compiler optimization (CWE-733) removes or modifies code used to detect integer overflow (CWE-190), allowing out-of-bounds write (CWE-787).

**•** 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-2010-2753: Chain: integer overflow leads to use-after-free

**•** CVE-2005-1513: Chain: integer overflow in securely-coded mail program leads to buffer overflow. In 2005, this was regarded as unrealistic to exploit, but in 2020, it was rediscovered to be easier to exploit due to evolutions of the technology.

**•** CVE-2002-0391: Integer overflow via a large number of arguments.

**•** CVE-2002-0639: Integer overflow in OpenSSH as listed in the demonstrative examples.

**•** CVE-2005-1141: Image with large width and height leads to integer overflow.

**•** CVE-2005-0102: Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.

**•** CVE-2004-2013: Length value of -1 leads to allocation of 0 bytes and resultant heap overflow.

**•** CVE-2017-1000121: chain: unchecked message size metadata allows integer overflow (CWE-190) leading to buffer overflow (CWE-119).

**•** CVE-2013-1591: Chain: an integer overflow (CWE-190) in the image size calculation causes an infinite loop (CWE-835) which sequentially allocates buffers without limits (CWE-1325) until the stack is full.

## Related Attack Patterns (CAPEC)

* CAPEC-92

## Modes of Introduction

**•** Implementation: This weakness may become security critical when determining the offset or size in behaviors such as memory allocation, copying, and concatenation.

## Common Consequences

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (Memory), DoS: Instability — Notes: This weakness can generally lead to undefined behavior and therefore crashes. When the calculated result is used for resource allocation, this weakness can cause too many (or too few) resources to be allocated, possibly enabling crashes if the product requests more resources than can be provided.

**•** Impact: Modify Memory — Notes: If the value in question is important to data (as opposed to flow), simple data corruption has occurred. Also, if the overflow/wraparound results in other conditions such as buffer overflows, further memory corruption may occur.

**•** Impact: Execute Unauthorized Code or Commands, Bypass Protection Mechanism — Notes: This weakness can sometimes trigger buffer overflows, which can be used to execute arbitrary code. This is usually outside the scope of the product's implicit security policy.

**•** Impact: Alter Execution Logic, DoS: Crash, Exit, or Restart, DoS: Resource Consumption (CPU) — Notes: If the overflow/wraparound occurs in a loop index variable, this could cause the loop to terminate at the wrong time - too early, too late, or not at all (i.e., infinite loops). With too many iterations, some loops could consume too many resources such as memory, file handles, etc., possibly leading to a crash or other DoS.

**•** Impact: Bypass Protection Mechanism — Notes: If integer values are used in security-critical decisions, such as calculating quotas or allocation limits, integer overflows can be used to cause an incorrect security decision.

## Potential Mitigations

**•** Requirements: Ensure that all protocols are strictly defined, such that all out-of-bounds behavior can be identified simply, and require strict conformance to the protocol. (Effectiveness: N/A)

**•** Requirements: Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. If possible, choose a language or compiler that performs automatic bounds checking. (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. Use libraries or frameworks that make it easier to handle numbers without unexpected consequences. Examples include safe integer handling packages such as SafeInt (C++) or IntegerLib (C or C++). [REF-106] (Effectiveness: N/A)

**•** Implementation: Perform input validation on any numeric input by ensuring that it is within the expected range. Enforce that the input meets both the minimum and maximum requirements for the expected range. Use unsigned integers where possible. This makes it easier to perform validation for integer overflows. When signed integers are required, ensure that the range check includes minimum values as well as maximum values. (Effectiveness: N/A)

**•** Implementation: Understand the programming language's underlying representation and how it interacts with numeric calculation (CWE-681). Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how the language handles numbers that are too large or too small for its underlying representation. [REF-7] Also be careful to account for 32-bit, 64-bit, and other potential differences that may affect the numeric representation. (Effectiveness: N/A)

**•** Architecture and Design: For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. (Effectiveness: N/A)

**•** Implementation: Examine compiler warnings closely and eliminate problems with potential security implications, such as signed / unsigned mismatch in memory operations, or use of uninitialized variables. Even if the weakness is rarely exploitable, a single failure may lead to the compromise of the entire system. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** This code intends to allocate a table of size num\_imgs, however as num\_imgs grows large, the calculation determining the size of the list will eventually overflow (CWE-190). This will result in a very small list to be allocated instead. If the subsequent code operates on the list as if it were num\_imgs long, it may result in many types of out-of-bounds problems (CWE-119).

**•** If nresp has the value 1073741824 and sizeof(char\*) has its typical value of 4, then the result of the operation nresp\*sizeof(char\*) overflows, and the argument to xmalloc() will be 0. Most malloc() implementations will happily allocate a 0-byte buffer, causing the subsequent loop iterations to overflow the heap buffer response.

**•** In the above case, it is entirely possible that bytesRec may overflow, continuously creating a lower number than MAXGET and also overwriting the first MAXGET-1 bytes of buf.

**•** However, in this example the primitive type short int is used for both the monthly and the quarterly sales variables. In C the short int primitive type has a maximum value of 32768. This creates a potential integer overflow if the value for the three monthly sales adds up to more than the maximum value for the short int primitive type. An integer overflow can lead to data corruption, unexpected behavior, infinite loops and system crashes. To correct the situation the appropriate primitive type should be used, as in the example below, and/or provide some validation mechanism to ensure that the maximum value for the primitive type is not exceeded.

## Notes

**•** Relationship: Integer overflows can be primary to buffer overflows when they cause less memory to be allocated than expected.

**•** Terminology: "Integer overflow" is
 sometimes used to cover several types of errors, including
 signedness errors, or buffer overflows that involve
 manipulation of integer data types instead of
 characters. Part of the confusion results from the fact
 that 0xffffffff is -1 in a signed context. Other confusion
 also arises because of the role that integer overflows
 have in chains. A "wraparound" is a well-defined, standard
 behavior that follows specific rules for how to handle
 situations when the intended numeric value is too large or
 too small to be represented, as specified in standards
 such as C11. "Overflow" is sometimes conflated with
 "wraparound" but typically indicates a non-standard or
 undefined behavior. The "overflow" term is sometimes used to indicate
 cases where either the maximum or the minimum is exceeded,
 but others might only use "overflow" to indicate exceeding
 the maximum while using "underflow" for exceeding the
 minimum. Some people use "overflow" to mean any value
 outside the representable range - whether greater than the
 maximum, or less than the minimum - but CWE uses
 "underflow" for cases in which the intended result is less
 than the minimum. See [REF-1440] for additional explanation of the
 ambiguity of terminology.

**•** Other: While there may be circumstances in
 which the logic intentionally relies on wrapping - such as
 with modular arithmetic in timers or counters - it can
 have security consequences if the wrap is unexpected.
 This is especially the case if the integer overflow can be
 triggered using user-supplied inputs.