# CWE Detail – CWE-191

## Description

The product subtracts one value from another, such that the result is less than the minimum allowable integer value, which produces a value that is not equal to the correct result.

## Extended Description

This can happen in signed and unsigned cases.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2004-0816: Integer underflow in firewall via malformed packet.

**•** CVE-2004-1002: Integer underflow by packet with invalid length.

**•** CVE-2005-0199: Long input causes incorrect length calculation.

**•** CVE-2005-1891: Malformed icon causes integer underflow in loop counter variable.

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (CPU), DoS: Resource Consumption (Memory), DoS: Instability — Notes: This weakness will generally lead to undefined behavior and therefore crashes. In the case of overflows involving loop index variables, the likelihood of infinite loops is also high.

**•** 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 wrap around 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 a program's implicit security policy.

## Applicable Platforms

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

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

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

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

## Demonstrative Examples

**•** The example has an integer underflow. The value of i is already at the lowest negative value possible, so after subtracting 1, the new value of i is 2147483647.

**•** 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.