# CWE Detail – CWE-682

## Description

The product performs a calculation that generates incorrect or unintended results that are later used in security-critical decisions or resource management.

## Extended Description

When product performs a security-critical calculation incorrectly, it might lead to incorrect resource allocations, incorrect privilege assignments, or failed comparisons among other things. Many of the direct results of an incorrect calculation can lead to even larger problems such as failed protection mechanisms or even arbitrary code execution.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2020-0022: chain: mobile phone Bluetooth implementation does not include offset when calculating packet length (CWE-682), leading to out-of-bounds write (CWE-787)

**•** CVE-2004-1363: substitution overflow: buffer overflow using environment variables that are expanded after the length check is performed

## Related Attack Patterns (CAPEC)

* CAPEC-128
* CAPEC-129

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Crash, Exit, or Restart — Notes: If the incorrect calculation causes the program to move into an unexpected state, it may lead to a crash or impairment of service.

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (Other), Execute Unauthorized Code or Commands — Notes: If the incorrect calculation is used in the context of resource allocation, it could lead to an out-of-bounds operation (CWE-119) leading to a crash or even arbitrary code execution. Alternatively, it may result in an integer overflow (CWE-190) and / or a resource consumption problem (CWE-400).

**•** Impact: Gain Privileges or Assume Identity — Notes: In the context of privilege or permissions assignment, an incorrect calculation can provide an attacker with access to sensitive resources.

**•** Impact: Bypass Protection Mechanism — Notes: If the incorrect calculation leads to an insufficient comparison (CWE-697), it may compromise a protection mechanism such as a validation routine and allow an attacker to bypass the security-critical code.

## Potential Mitigations

**•** Implementation: Understand your programming language's underlying representation and how it interacts with numeric calculation. Pay close attention to byte size discrepancies, precision, signed/unsigned distinctions, truncation, conversion and casting between types, "not-a-number" calculations, and how your language handles numbers that are too large or too small for its underlying representation. (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. (Effectiveness: N/A)

**•** Implementation: Use the appropriate type for the desired action. For example, in C/C++, only use unsigned types for values that could never be negative, such as height, width, or other numbers related to quantity. (Effectiveness: N/A)

**•** Architecture and Design: Use languages, 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++). (Effectiveness: N/A)

**•** Architecture and Design: Use languages, 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++). (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)

**•** Testing: Use automated static analysis tools that target this type of weakness. Many modern techniques use data flow analysis to minimize the number of false positives. This is not a perfect solution, since 100% accuracy and coverage are not feasible. (Effectiveness: N/A)

**•** Testing: Use dynamic tools and techniques that interact with the product using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The product's operation may slow down, but it should not become unstable, crash, or generate incorrect results. (Effectiveness: N/A)

## Applicable Platforms

**•** 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).

**•** The code does not consider the event that the team they are querying has not scored a touchdown, but has gained yardage. In that case, we should expect an ArithmeticException to be thrown by the JVM. This could lead to a loss of availability if our error handling code is not set up correctly.

**•** In this example, second\_char is intended to point to the second byte of p. But, adding 1 to p actually adds sizeof(int) to p, giving a result that is incorrect (3 bytes off on 32-bit platforms). If the resulting memory address is read, this could potentially be an information leak. If it is a write, it could be a security-critical write to unauthorized memory-- whether or not it is a buffer overflow. Note that the above code may also be wrong in other ways, particularly in a little endian environment.

## Notes

**•** Research Gap: Weaknesses related to this Pillar appear to be under-studied, especially with respect to classification schemes. Input from academic and other communities could help identify and resolve gaps or organizational difficulties within CWE.