# CWE Detail – CWE-468

## Description

In C and C++, one may often accidentally refer to the wrong memory due to the semantics of when math operations are implicitly scaled.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Modes of Introduction

**•** Implementation: Programmers may try to index from a pointer by adding a number of bytes. This is incorrect because C and C++ implicitly scale the operand by the size of the data type.

## Common Consequences

**•** Impact: Read Memory, Modify Memory — Notes: Incorrect pointer scaling will often result in buffer overflow conditions. Confidentiality can be compromised if the weakness is in the context of a buffer over-read or under-read.

## Potential Mitigations

**•** Architecture and Design: Use a platform with high-level memory abstractions. (Effectiveness: N/A)

**•** Implementation: Always use array indexing instead of direct pointer manipulation. (Effectiveness: N/A)

**•** Architecture and Design: Use technologies for preventing buffer overflows. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** 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.