# CWE Detail – CWE-123

## Description

Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.5

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** 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.

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Modify Memory, Execute Unauthorized Code or Commands, Gain Privileges or Assume Identity, DoS: Crash, Exit, or Restart, Bypass Protection Mechanism — Notes: Clearly, write-what-where conditions can be used to write data to areas of memory outside the scope of a policy. Also, they almost invariably can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. If the attacker can overwrite a pointer's worth of memory (usually 32 or 64 bits), they can redirect a function pointer to their own malicious code. Even when the attacker can only modify a single byte arbitrary code execution can be possible. Sometimes this is because the same problem can be exploited repeatedly to the same effect. Other times it is because the attacker can overwrite security-critical application-specific data -- such as a flag indicating whether the user is an administrator.

**•** Impact: DoS: Crash, Exit, or Restart, Modify Memory — Notes: Many memory accesses can lead to program termination, such as when writing to addresses that are invalid for the current process.

**•** Impact: Bypass Protection Mechanism, Other — Notes: When the consequence is arbitrary code execution, this can often be used to subvert any other security service.

## Potential Mitigations

**•** Architecture and Design: Use a language that provides appropriate memory abstractions. (Effectiveness: N/A)

**•** Operation: Use OS-level preventative functionality integrated after the fact. Not a complete solution. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** Vulnerability in this case is dependent on memory layout. The call to strcpy() can be used to write past the end of buf1, and, with a typical layout, can overwrite the accounting information that the system keeps for buf2 when it is allocated. Note that if the allocation header for buf2 can be overwritten, buf2 itself can be overwritten as well.