# CWE Detail – CWE-825

## Description

The product dereferences a pointer that contains a location for memory that was previously valid, but is no longer valid.

## Extended Description

When a product releases memory, but it maintains a pointer to that memory, then the memory might be re-allocated at a later time. If the original pointer is accessed to read or write data, then this could cause the product to read or modify data that is in use by a different function or process. Depending on how the newly-allocated memory is used, this could lead to a denial of service, information exposure, or code execution.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2008-5013: access of expired memory address leads to arbitrary code execution

**•** CVE-2010-3257: stale pointer issue leads to denial of service and possibly other consequences

**•** CVE-2008-0062: Chain: a message having an unknown message type may cause a reference to uninitialized memory resulting in a null pointer dereference (CWE-476) or dangling pointer (CWE-825), possibly crashing the system or causing heap corruption.

**•** CVE-2007-1211: read of value at an offset into a structure after the offset is no longer valid

## Common Consequences

**•** Impact: Read Memory — Notes: If the expired pointer is used in a read operation, an attacker might be able to control data read in by the application.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: If the expired pointer references a memory location that is not accessible to the product, or points to a location that is "malformed" (such as NULL) or larger than expected by a read or write operation, then a crash may occur.

**•** Impact: Execute Unauthorized Code or Commands — Notes: If the expired pointer is used in a function call, or points to unexpected data in a write operation, then code execution may be possible.

## Potential Mitigations

**•** Architecture and Design: Choose a language that provides automatic memory management. (Effectiveness: N/A)

**•** Implementation: When freeing pointers, be sure to set them to NULL once they are freed. However, the utilization of multiple or complex data structures may lower the usefulness of this strategy. (Effectiveness: N/A)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** When an error occurs, the pointer is immediately freed. However, this pointer is later incorrectly used in the logError function.

**•** Double free vulnerabilities have two common (and sometimes overlapping) causes:

## Notes

**•** Maintenance: There are close relationships between incorrect pointer dereferences and other weaknesses related to buffer operations. There may not be sufficient community agreement regarding these relationships. Further study is needed to determine when these relationships are chains, composites, perspective/layering, or other types of relationships. As of September 2010, most of the relationships are being captured as chains.

**•** Terminology: Many weaknesses related to pointer dereferences fall under the general term of "memory corruption" or "memory safety." As of September 2010, there is no commonly-used terminology that covers the lower-level variants.