# CWE Detail – CWE-479

## Description

The product defines a signal handler that calls a non-reentrant function.

## Extended Description

Non-reentrant functions are functions that cannot safely be called, interrupted, and then recalled before the first call has finished without resulting in memory corruption. This can lead to an unexpected system state and unpredictable results with a variety of potential consequences depending on context, including denial of service and code execution. Many functions are not reentrant, but some of them can result in the corruption of memory if they are used in a signal handler. The function call syslog() is an example of this. In order to perform its functionality, it allocates a small amount of memory as "scratch space." If syslog() is suspended by a signal call and the signal handler calls syslog(), the memory used by both of these functions enters an undefined, and possibly, exploitable state. Implementations of malloc() and free() manage metadata in global structures in order to track which memory is allocated versus which memory is available, but they are non-reentrant. Simultaneous calls to these functions can cause corruption of the metadata.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2005-0893: signal handler calls function that ultimately uses malloc()

**•** CVE-2004-2259: SIGCHLD signal to FTP server can cause crash under heavy load while executing non-reentrant functions like malloc/free.

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes: It may be possible to execute arbitrary code through the use of a write-what-where condition.

**•** Impact: Modify Memory, Modify Application Data — Notes: Signal race conditions often result in data corruption.

## Potential Mitigations

**•** Requirements: Require languages or libraries that provide reentrant functionality, or otherwise make it easier to avoid this weakness. (Effectiveness: N/A)

**•** Architecture and Design: Design signal handlers to only set flags rather than perform complex functionality. (Effectiveness: N/A)

**•** Implementation: Ensure that non-reentrant functions are not found in signal handlers. (Effectiveness: N/A)

**•** Implementation: Use sanity checks to reduce the timing window for exploitation of race conditions. This is only a partial solution, since many attacks might fail, but other attacks still might work within the narrower window, even accidentally. (Effectiveness: Defense in Depth)

## Applicable Platforms

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

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

## Demonstrative Examples

**•** N/A