# CWE Detail – CWE-367

## Description

The product checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the product to perform invalid actions when the resource is in an unexpected state.

## Extended Description

This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2015-1743: TOCTOU in sandbox process allows installation of untrusted browser add-ons by replacing a file after it has been verified, but before it is executed

**•** CVE-2003-0813: Chain: A multi-threaded race condition (CWE-367) allows attackers to cause two threads to process the same RPC request, which causes a use-after-free (CWE-416) in one thread

**•** CVE-2004-0594: PHP flaw allows remote attackers to execute arbitrary code by aborting execution before the initialization of key data structures is complete.

**•** CVE-2008-2958: chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.

**•** CVE-2008-1570: chain: time-of-check time-of-use (TOCTOU) race condition in program allows bypass of protection mechanism that was designed to prevent symlink attacks.

## Related Attack Patterns (CAPEC)

* CAPEC-27
* CAPEC-29

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Alter Execution Logic, Unexpected State — Notes: The attacker can gain access to otherwise unauthorized resources.

**•** Impact: Modify Application Data, Modify Files or Directories, Modify Memory, Other — Notes: Race conditions such as this kind may be employed to gain read or write access to resources which are not normally readable or writable by the user in question.

**•** Impact: Other — Notes: The resource in question, or other resources (through the corrupted one), may be changed in undesirable ways by a malicious user.

**•** Impact: Hide Activities — Notes: If a file or other resource is written in this method, as opposed to in a valid way, logging of the activity may not occur.

**•** Impact: Other — Notes: In some cases it may be possible to delete files a malicious user might not otherwise have access to, such as log files.

## Potential Mitigations

**•** Implementation: The most basic advice for TOCTOU vulnerabilities is to not perform a check before the use. This does not resolve the underlying issue of the execution of a function on a resource whose state and identity cannot be assured, but it does help to limit the false sense of security given by the check. (Effectiveness: N/A)

**•** Implementation: When the file being altered is owned by the current user and group, set the effective gid and uid to that of the current user and group when executing this statement. (Effectiveness: N/A)

**•** Architecture and Design: Limit the interleaving of operations on files from multiple processes. (Effectiveness: N/A)

**•** Implementation: If you cannot perform operations atomically and you must share access to the resource between multiple processes or threads, then try to limit the amount of time (CPU cycles) between the check and use of the resource. This will not fix the problem, but it could make it more difficult for an attack to succeed. (Effectiveness: N/A)

**•** Implementation: Recheck the resource after the use call to verify that the action was taken appropriately. (Effectiveness: N/A)

**•** Architecture and Design: Ensure that some environmental locking mechanism can be used to protect resources effectively. (Effectiveness: N/A)

**•** Implementation: Ensure that locking occurs before the check, as opposed to afterwards, such that the resource, as checked, is the same as it is when in use. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** Potentially the file could have been updated between the time of the check and the lstat, especially since the printf has latency.

**•** The call to access() behaves as expected, and returns 0 if the user running the program has the necessary permissions to write to the file, and -1 otherwise. However, because both access() and fopen() operate on filenames rather than on file handles, there is no guarantee that the file variable still refers to the same file on disk when it is passed to fopen() that it did when it was passed to access(). If an attacker replaces file after the call to access() with a symbolic link to a different file, the program will use its root privileges to operate on the file even if it is a file that the attacker would otherwise be unable to modify. By tricking the program into performing an operation that would otherwise be impermissible, the attacker has gained elevated privileges. This type of vulnerability is not limited to programs with root privileges. If the application is capable of performing any operation that the attacker would not otherwise be allowed perform, then it is a possible target.

**•** This code attempts to resolve symbolic links before checking the file and printing its contents. However, an attacker may be able to change the file from a real file to a symbolic link between the calls to is\_link() and file\_get\_contents(), allowing the reading of arbitrary files. Note that this code fails to log the attempted access (CWE-778).

**•** If this code block were executed within multiple threads, and MYFILE.txt changed between the operation of one thread and another, then the switch could produce different, possibly unexpected results.

## Notes

**•** Relationship: TOCTOU issues do not always involve symlinks, and not every symlink issue is a TOCTOU problem.

**•** Research Gap: Non-symlink TOCTOU issues are not reported frequently, but they are likely to occur in code that attempts to be secure.