# CWE Detail – CWE-667

## Description

The product does not properly acquire or release a lock on a resource, leading to unexpected resource state changes and behaviors.

## Extended Description

Locking is a type of synchronization behavior that ensures that multiple independently-operating processes or threads do not interfere with each other when accessing the same resource. All processes/threads are expected to follow the same steps for locking. If these steps are not followed precisely - or if no locking is done at all - then another process/thread could modify the shared resource in a way that is not visible or predictable to the original process. This can lead to data or memory corruption, denial of service, etc.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2021-1782: Chain: improper locking (CWE-667) leads to race condition (CWE-362), as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2009-0935: Attacker provides invalid address to a memory-reading function, causing a mutex to be unlocked twice

**•** CVE-2010-4210: function in OS kernel unlocks a mutex that was not previously locked, causing a panic or overwrite of arbitrary memory.

**•** CVE-2008-4302: Chain: OS kernel does not properly handle a failure of a function call (CWE-755), leading to an unlock of a resource that was not locked (CWE-832), with resultant crash.

**•** CVE-2009-1243: OS kernel performs an unlock in some incorrect circumstances, leading to panic.

**•** CVE-2009-2857: OS deadlock

**•** CVE-2009-1961: OS deadlock involving 3 separate functions

**•** CVE-2009-2699: deadlock in library

**•** CVE-2009-4272: deadlock triggered by packets that force collisions in a routing table

**•** CVE-2002-1850: read/write deadlock between web server and script

**•** CVE-2004-0174: web server deadlock involving multiple listening connections

**•** CVE-2009-1388: multiple simultaneous calls to the same function trigger deadlock.

**•** CVE-2006-5158: chain: other weakness leads to NULL pointer dereference (CWE-476) or deadlock (CWE-833).

**•** CVE-2006-4342: deadlock when an operation is performed on a resource while it is being removed.

**•** CVE-2006-2374: Deadlock in device driver triggered by using file handle of a related device.

**•** CVE-2006-2275: Deadlock when large number of small messages cannot be processed quickly enough.

**•** CVE-2005-3847: OS kernel has deadlock triggered by a signal during a core dump.

**•** CVE-2005-3106: Race condition leads to deadlock.

**•** CVE-2005-2456: Chain: array index error (CWE-129) leads to deadlock (CWE-833)

**•** CVE-2001-0682: Program can not execute when attacker obtains a mutex.

**•** CVE-2002-1914: Program can not execute when attacker obtains a lock on a critical output file.

**•** CVE-2002-1915: Program can not execute when attacker obtains a lock on a critical output file.

**•** CVE-2002-0051: Critical file can be opened with exclusive read access by user, preventing application of security policy. Possibly related to improper permissions, large-window race condition.

**•** CVE-2000-0338: Chain: predictable file names used for locking, allowing attacker to create the lock beforehand. Resultant from permissions and randomness.

**•** CVE-2000-1198: Chain: Lock files with predictable names. Resultant from randomness.

**•** CVE-2002-1869: Product does not check if it can write to a log file, allowing attackers to avoid logging by accessing the file using an exclusive lock. Overlaps unchecked error condition. This is not quite CWE-412, but close.

## Related Attack Patterns (CAPEC)

* CAPEC-25
* CAPEC-26
* CAPEC-27

## Attack TTPs

**•** T1499.004: Application or System Exploitation (Tactics: impact)

## Modes of Introduction

**•** Architecture and Design: N/A

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Resource Consumption (CPU) — Notes: Inconsistent locking discipline can lead to deadlock.

## Potential Mitigations

**•** Implementation: Use industry standard APIs to implement locking mechanism. (Effectiveness: N/A)

## Demonstrative Examples

**•** N/A

**•** PHP by default will wait indefinitely until a file lock is released. If an attacker is able to obtain the file lock, this code will pause execution, possibly leading to denial of service for other users. Note that in this case, if an attacker can perform an flock() on the file, they may already have privileges to destroy the log file. However, this still impacts the execution of other programs that depend on flock().

**•** However, the code does not check the value returned by pthread\_mutex\_lock() for errors. If pthread\_mutex\_lock() cannot acquire the mutex for any reason, the function may introduce a race condition into the program and result in undefined behavior.

**•** The programmer wants to guarantee that only one Helper() object is ever allocated, but does not want to pay the cost of synchronization every time this code is called.

## Notes

**•** Maintenance: Deeper research is necessary for synchronization and related mechanisms, including locks, mutexes, semaphores, and other mechanisms. Multiple entries are dependent on this research, which includes relationships to concurrency, race conditions, reentrant functions, etc. CWE-662 and its children - including CWE-667, CWE-820, CWE-821, and others - may need to be modified significantly, along with their relationships.