# CWE Detail – CWE-662

## Description

The product utilizes multiple threads or processes to allow temporary access to a shared resource that can only be exclusive to one process at a time, but it does not properly synchronize these actions, which might cause simultaneous accesses of this resource by multiple threads or processes.

## Extended Description

Synchronization refers to a variety of behaviors and mechanisms that allow two or more independently-operating processes or threads to ensure that they operate on shared resources in predictable ways that do not interfere with each other. Some shared resource operations cannot be executed atomically; that is, multiple steps must be guaranteed to execute sequentially, without any interference by other processes. Synchronization mechanisms vary widely, but they may include locking, mutexes, and semaphores. When a multi-step operation on a shared resource cannot be guaranteed to execute independent of interference, then the resulting behavior can be unpredictable. Improper synchronization could 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

## Related Attack Patterns (CAPEC)

* CAPEC-25
* CAPEC-26
* CAPEC-27
* CAPEC-29

## Attack TTPs

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

## Modes of Introduction

**•** Architecture and Design: N/A

**•** Implementation: N/A

## Common Consequences

**•** Impact: Modify Application Data, Read Application Data, Alter Execution Logic — Notes:

## Potential Mitigations

**•** Implementation: Use industry standard APIs to synchronize your code. (Effectiveness: N/A)

## Demonstrative Examples

**•** 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.

**•** One might expect the code to print out something like:

## 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.