# CWE Detail – CWE-412

## Description

The product properly checks for the existence of a lock, but the lock can be externally controlled or influenced by an actor that is outside of the intended sphere of control.

## Extended Description

This prevents the product from acting on associated resources or performing other behaviors that are controlled by the presence of the lock. Relevant locks might include an exclusive lock or mutex, or modifying a shared resource that is treated as a lock. If the lock can be held for an indefinite period of time, then the denial of service could be permanent.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** 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

## 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 (Other) — Notes: When an attacker can control a lock, the program may wait indefinitely until the attacker releases the lock, causing a denial of service to other users of the program. This is especially problematic if there is a blocking operation on the lock.

## Potential Mitigations

**•** Architecture and Design: Use any access control that is offered by the functionality that is offering the lock. (Effectiveness: N/A)

**•** Architecture and Design: Use unpredictable names or identifiers for the locks. This might not always be possible or feasible. (Effectiveness: N/A)

**•** Architecture and Design: Consider modifying your code to use non-blocking synchronization methods. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** 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().

## Notes

**•** Relationship: This overlaps Insufficient Resource Pool when the "pool" is of size 1. It can also be resultant from race conditions, although the timing window could be quite large in some cases.