# CWE Detail – CWE-404

## Description

The product does not release or incorrectly releases a resource before it is made available for re-use.

## Extended Description

When a resource is created or allocated, the developer is responsible for properly releasing the resource as well as accounting for all potential paths of expiration or invalidation, such as a set period of time or revocation.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-1999-1127: Does not shut down named pipe connections if malformed data is sent.

**•** CVE-2001-0830: Sockets not properly closed when attacker repeatedly connects and disconnects from server.

**•** CVE-2002-1372: Chain: Return values of file/socket operations are not checked (CWE-252), allowing resultant consumption of file descriptors (CWE-772).

## Related Attack Patterns (CAPEC)

* CAPEC-125
* CAPEC-130
* CAPEC-131
* CAPEC-494
* CAPEC-495
* CAPEC-496
* CAPEC-666

## Attack TTPs

**•** T1498.001: Direct Network Flood (Tactics: impact)

**•** T1499.001: OS Exhaustion Flood (Tactics: impact)

**•** T1499.003: Application Exhaustion Flood (Tactics: impact)

**•** T1499: Endpoint Denial of Service (Tactics: impact)

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Resource Consumption (Other), Varies by Context — Notes: Most unreleased resource issues result in general software reliability problems, but if an attacker can intentionally trigger a resource leak, the attacker might be able to launch a denial of service attack by depleting the resource pool.

**•** Impact: Read Application Data — Notes: When a resource containing sensitive information is not correctly shutdown, it may expose the sensitive data in a subsequent allocation.

## Potential Mitigations

**•** Requirements: Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, languages such as Java, Ruby, and Lisp perform automatic garbage collection that releases memory for objects that have been deallocated. (Effectiveness: N/A)

**•** Implementation: It is good practice to be responsible for freeing all resources you allocate and to be consistent with how and where you free memory in a function. If you allocate memory that you intend to free upon completion of the function, you must be sure to free the memory at all exit points for that function including error conditions. (Effectiveness: N/A)

**•** Implementation: Memory should be allocated/freed using matching functions such as malloc/free, new/delete, and new[]/delete[]. (Effectiveness: N/A)

**•** Implementation: When releasing a complex object or structure, ensure that you properly dispose of all of its member components, not just the object itself. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** The good code example simply adds an explicit call to the Close() function when the system is done using the file. Within a simple example such as this the problem is easy to see and fix. In a real system, the problem may be considerably more obscure.

**•** If an exception occurs after establishing the database connection and before the same connection closes, the pool of database connections may become exhausted. If the number of available connections is exceeded, other users cannot access this resource, effectively denying access to the application.

**•** N/A

**•** N/A

**•** N/A

**•** N/A

## Notes

**•** Relationship: Overlaps memory leaks, asymmetric resource consumption, malformed input errors.