# CWE Detail – CWE-772

## Description

The product does not release a resource after its effective lifetime has ended, i.e., after the resource is no longer needed.

## Extended Description

When a resource is not released after use, it can allow attackers to cause a denial of service by causing the allocation of resources without triggering their release. Frequently-affected resources include memory, CPU, disk space, power or battery, etc.

## Threat-Mapped Scoring

Score: 1.9

Priority: P3 - Important (Medium)

## Observed Examples (CVEs)

**•** CVE-2007-0897: Chain: anti-virus product encounters a malformed file but returns from a function without closing a file descriptor (CWE-775) leading to file descriptor consumption (CWE-400) and failed scans.

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

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

**•** CVE-2009-2858: Chain: memory leak (CWE-404) leads to resource exhaustion.

**•** CVE-2009-2054: Product allows exhaustion of file descriptors when processing a large number of TCP packets.

**•** CVE-2008-2122: Port scan triggers CPU consumption with processes that attempt to read data from closed sockets.

**•** CVE-2007-4103: Product allows resource exhaustion via a large number of calls that do not complete a 3-way handshake.

**•** 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-469

## Attack TTPs

**•** T1499.002: Service Exhaustion Flood (Tactics: impact)

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: DoS: Resource Consumption (Other) — Notes: An attacker that can influence the allocation of resources that are not properly released could deplete the available resource pool and prevent all other processes from accessing the same type of resource.

## 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 resources in a function. If you allocate resources that you intend to free upon completion of the function, you must be sure to free the resources at all exit points for that function including error conditions. (Effectiveness: N/A)

**•** Operation: Use resource-limiting settings provided by the operating system or environment. For example, when managing system resources in POSIX, setrlimit() can be used to set limits for certain types of resources, and getrlimit() can determine how many resources are available. However, these functions are not available on all operating systems. When the current levels get close to the maximum that is defined for the application (see CWE-770), then limit the allocation of further resources to privileged users; alternately, begin releasing resources for less-privileged users. While this mitigation may protect the system from attack, it will not necessarily stop attackers from adversely impacting other users. Ensure that the application performs the appropriate error checks and error handling in case resources become unavailable (CWE-703). (Effectiveness: N/A)

## 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.

**•** The problem with the above code is that if an exception occurs while executing the SQL or processing the results, the SqlConnection object is not closed. If this happens often enough, the database will run out of available cursors and not be able to execute any more SQL queries.

**•** 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

## Notes

**•** Maintenance: "Resource exhaustion" (CWE-400) is currently treated as a weakness, although it is more like a category of weaknesses that all have the same type of consequence. While this entry treats CWE-400 as a parent in view 1000, the relationship is probably more appropriately described as a chain.

**•** Theoretical: Vulnerability theory is largely about how behaviors and resources interact. "Resource exhaustion" can be regarded as either a consequence or an attack, depending on the perspective. This entry is an attempt to reflect one of the underlying weaknesses that enable these attacks (or consequences) to take place.