# CWE Detail – CWE-400

## Description

The product does not properly control the allocation and maintenance of a limited resource.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2022-21668: Chain: Python library does not limit the resources used to process images that specify a very large number of bands (CWE-1284), leading to excessive memory consumption (CWE-789) or an integer overflow (CWE-190).

**•** CVE-2020-7218: Go-based workload orchestrator does not limit resource usage with unauthenticated connections, allowing a DoS by flooding the service

**•** CVE-2020-3566: Resource exhaustion in distributed OS because of "insufficient" IGMP queue management, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2009-2874: Product allows attackers to cause a crash via a large number of connections.

**•** CVE-2009-1928: Malformed request triggers uncontrolled recursion, leading to stack exhaustion.

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

**•** CVE-2009-2726: Driver does not use a maximum width when invoking sscanf style functions, causing stack consumption.

**•** CVE-2009-2540: Large integer value for a length property in an object causes a large amount of memory allocation.

**•** CVE-2009-2299: Web application firewall consumes excessive memory when an HTTP request contains a large Content-Length value but no POST data.

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

**•** CVE-2008-5180: Communication product allows memory consumption with a large number of SIP requests, which cause many sessions to be created.

**•** CVE-2008-2121: TCP implementation allows attackers to consume CPU and prevent new connections using a TCP SYN flood attack.

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

**•** CVE-2008-1700: Product allows attackers to cause a denial of service via a large number of directives, each of which opens a separate window.

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

**•** CVE-2006-1173: Mail server does not properly handle deeply nested multipart MIME messages, leading to stack exhaustion.

**•** 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.

## Related Attack Patterns (CAPEC)

* CAPEC-147
* CAPEC-227
* CAPEC-492

## Attack TTPs

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

## Modes of Introduction

**•** Operation: The product could be operated in a system or environment with lower resource limits than expected, which might make it easier for attackers to consume all available resources.

**•** System Configuration: The product could be configured with lower resource limits than expected, which might make it easier for attackers to consume all available resources.

**•** Architecture and Design: The designer might not consider how to handle and throttle excessive resource requests, which typically requires careful planning to handle more gracefully than a crash or exit.

**•** Implementation: There are at least three distinct scenarios that can commonly lead to resource exhaustion: Lack of throttling for the number of allocated resources. Losing all references to a resource before reaching the shutdown stage. Not closing/returning a resource after processing. Resource exhaustion problems often occur due to an incorrect implementation of the following situations: Error conditions and other exceptional circumstances. Confusion over which part of the program is responsible for releasing the resource.

## Common Consequences

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (CPU), DoS: Resource Consumption (Memory), DoS: Resource Consumption (Other) — Notes: If an attacker can trigger the allocation of the limited resources, but the number or size of the resources is not controlled, then the most common result is denial of service. This would prevent valid users from accessing the product, and it could potentially have an impact on the surrounding environment, i.e., the product may slow down, crash due to unhandled errors, or lock out legitimate users. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system.

**•** Impact: Bypass Protection Mechanism, Other — Notes: In some cases it may be possible to force the product to "fail open" in the event of resource exhaustion. The state of the product -- and possibly the security functionality - may then be compromised.

## Potential Mitigations

**•** Architecture and Design: Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold. (Effectiveness: N/A)

**•** Architecture and Design: Mitigation of resource exhaustion attacks requires that the target system either: recognizes the attack and denies that user further access for a given amount of time, or uniformly throttles all requests in order to make it more difficult to consume resources more quickly than they can again be freed. The first of these solutions is an issue in itself though, since it may allow attackers to prevent the use of the system by a particular valid user. If the attacker impersonates the valid user, they may be able to prevent the user from accessing the server in question. The second solution is simply difficult to effectively institute -- and even when properly done, it does not provide a full solution. It simply makes the attack require more resources on the part of the attacker. (Effectiveness: N/A)

**•** Architecture and Design: Ensure that protocols have specific limits of scale placed on them. (Effectiveness: N/A)

**•** Implementation: Ensure that all failures in resource allocation place the system into a safe posture. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** There are no limits to runnables. Potentially an attacker could cause resource problems very quickly.

**•** The program does not track how many connections have been made, and it does not limit the number of connections. Because forking is a relatively expensive operation, an attacker would be able to cause the system to run out of CPU, processes, or memory by making a large number of connections. Alternatively, an attacker could consume all available connections, preventing others from accessing the system remotely.

**•** This example creates a situation where data can be dumped to a file on the local file system without any limits on the size of the file. This could potentially exhaust file or disk resources and/or limit other clients' ability to access the service.

**•** This example creates a situation where the length of the body character array can be very large and will consume excessive memory, exhausting system resources. This can be avoided by restricting the length of the second character array with a maximum length check

**•** In this example there is no limit to the number of client connections and client threads that are created. Allowing an unlimited number of client connections and threads could potentially overwhelm the system and system resources.

**•** Because ReadAll is defined to read from src until EOF, it does not treat an EOF from Read as an error to be reported. This example creates a situation where the length of the body supplied can be very large and will consume excessive memory, exhausting system resources. This can be avoided by ensuring the body does not exceed a predetermined length of bytes.

## Notes

**•** Maintenance: "Resource consumption" could be interpreted as a consequence instead of an insecure behavior, so this entry is being considered for modification. It appears to be referenced too frequently when more precise mappings are available. Some of its children, such as CWE-771, might be better considered 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 the underlying weaknesses that enable these attacks (or consequences) to take place.

**•** Other: Database queries that take a long time to process are good DoS targets. An attacker would have to write a few lines of Perl code to generate enough traffic to exceed the site's ability to keep up. This would effectively prevent authorized users from using the site at all. Resources can be exploited simply by ensuring that the target machine must do much more work and consume more resources in order to service a request than the attacker must do to initiate a request. A prime example of this can be found in old switches that were vulnerable to "macof" attacks (so named for a tool developed by Dugsong). These attacks flooded a switch with random IP and MAC address combinations, therefore exhausting the switch's cache, which held the information of which port corresponded to which MAC addresses. Once this cache was exhausted, the switch would fail in an insecure way and would begin to act simply as a hub, broadcasting all traffic on all ports and allowing for basic sniffing attacks. Limited resources include memory, file system storage, database connection pool entries, CPU, and others.

**•** Maintenance: The Taxonomy\_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.