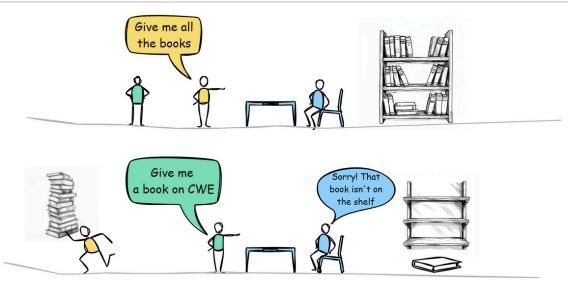
CWE-400: Uncontrolled Resource Consumption (4.19.1)

Archived: 2026-04-06 01:16:47 UTC

Weakness ID: 400

Vulnerability Mapping: DISCOURAGED This CWE ID should not be used to map to real-world vulnerabilities
Abstraction: Class Class - a weakness that is described in a very abstract fashion, typically independent of any specific language or technology. More specific than a Pillar Weakness, but more general than a Base Weakness. Class level weaknesses typically describe issues in terms of 1 or 2 of the following dimensions: behavior, property, and resource.

Description

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

Alternate Terms

Resource Exhaustion	

Common Consequences

This table specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Impact	Details
DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory); DoS: Resource Consumption (Other)	Scope: Availability 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.
Bypass Protection Mechanism; Other	Scope: Access Control, Other 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

Phase(s)	Mitigation
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.
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.
Architecture and Design	Ensure that protocols have specific limits of scale placed on them.
Implementation	Ensure that all failures in resource allocation place the system into a safe posture.

Relationships

This table shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

Relevant to the view "Research Concepts" (View-1000)

Nature	Type
ChildOf	Pillar - a weakness that is the most abstract type of weakness and represents a theme for all class/base/variant weaknesses related to it
ParentOf	Class - a weakness that is described in a very abstract fashion, typically independent of any specific language or technology. More spe
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f

ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
CanFollow	Class - a weakness that is described in a very abstract fashion, typically independent of any specific language or technology. More spe

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (View-1003)

Nature	Type
MemberOf	View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f
ParentOf	Base - a weakness that is still mostly independent of a resource or technology, but with sufficient details to provide specific methods f

Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

Phase	Note
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.

Applicable Platforms

This listing shows possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages	Class: Not Language-Specific (Undetermined Prevalence)
Technologies	Class: Not Technology-Specific (Undetermined Prevalence)

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

The following example demonstrates the weakness.

(bad code)

Example Language: Java

```
class Worker implements Executor {
...
public void execute(Runnable r) {
try {
...
}
catch (InterruptedException ie) {
// postpone response
Thread.currentThread().interrupt();
}
}
public Worker(Channel ch, int nworkers) {
...
}
protected void activate() {
Runnable loop = new Runnable() {
public void run() {
try {
for (;;) {
Runnable r = ...;
r.run();
}
}
catch (InterruptedException ie) {
...
}
}
```

```
};
new Thread(loop).start();

}

}
```

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

Example 2

This code allocates a socket and forks each time it receives a new connection.

(bad code)

Example Language: C

```
sock=socket(AF_INET, SOCK_STREAM, 0);
while (1) {

newsock=accept(sock, ...);
printf("A connection has been accepted\n");
pid = fork();

}
```

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.

Example 3

In the following example a server socket connection is used to accept a request to store data on the local file system using a specified filename. The method openSocketConnection establishes a server socket to accept requests from a client. When a client establishes a connection to this service the getNextMessage method is first used to retrieve from the socket the name of the file to store the data, the openFileToWrite method will validate the filename and open a file to write to on the local file system. The getNextMessage is then used within a while loop to continuously read data from the socket and output the data to the file until there is no longer any data from the socket.

(bad code)

Example Language: C

```
int writeDataFromSocketToFile(char *host, int port)
{

char filename[FILENAME_SIZE];
char buffer[BUFFER_SIZE];
int socket = openSocketConnection(host, port);

if (socket < 0) {

printf("Unable to open socket connection");
return(FAIL);

}
if (getNextMessage(socket, filename, FILENAME_SIZE) > 0) {

if (openFileToWrite(filename) > 0) {

while (getNextMessage(socket, buffer, BUFFER_SIZE) > 0){

if (!(writeToFile(buffer) > 0))

break;

}
```

```
}
closeFile();

}
closeSocket(socket);

}
```

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.

Example 4

In the following example, the processMessage method receives a two dimensional character array containing the message to be processed. The two-dimensional character array contains the length of the message in the first character array and the message body in the second character array. The getMessageLength method retrieves the integer value of the length from the first character array. After validating that the message length is greater than zero, the body character array pointer points to the start of the second character array of the two-dimensional character array and memory is allocated for the new body character array.

(bad code)

Example Language: C

```
/* process message accepts a two-dimensional character array of the form [length][body] containing the message to be processed */
int processMessage(char **message)
{

char *body;

int length = getMessageLength(message[0]);

if (length > 0) {

body = &message[1][0];
processMessageBody(body);
return(SUCCESS);

}
else {

printf("Unable to process message; invalid message length");
return(FAIL);

}

}
```

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

Also, consider changing the type from 'int' to 'unsigned int', so that you are always guaranteed that the number is positive. This might not be possible if the protocol specifically requires allowing negative values, or if you cannot control the return value from getMessageLength(), but it could simplify the check to ensure the input is positive, and eliminate other errors such as signed-to-unsigned conversion errors (CWE-195) that may occur elsewhere in the code.

(good code)

Example Language: C

```
unsigned int length = getMessageLength(message[0]);
if ((length > 0) && (length < MAX_LENGTH)) {...}
```

Example 5

In the following example, a server object creates a server socket and accepts client connections to the socket. For every client connection to the socket a separate thread object is generated using the ClientSocketThread class that handles request

made by the client through the socket.

(bad code)

Example Language: Java

```
public void acceptConnections() {

try {

ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
int counter = 0;
boolean hasConnections = true;
while (hasConnections) {

Socket client = serverSocket.accept();
Thread t = new Thread(new ClientSocketThread(client));
t.setName(client.getInetAddress().getHostName() + ":" + counter++);
t.start();

}
serverSocket.close();

} catch (IOException ex) {...}

}
```

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.

The server should limit the number of client connections and the client threads that are created. This can be easily done by creating a thread pool object that limits the number of threads that are generated.

(good code)

Example Language: Java

```
public static final int SERVER_PORT = 4444;
public static final int MAX_CONNECTIONS = 10;
...

public void acceptConnections() {

try {

ServerSocket serverSocket = new ServerSocket(SERVER_PORT);
int counter = 0;
boolean hasConnections = true;
while (hasConnections) {

hasConnections = checkForMoreConnections();
Socket client = serverSocket.accept();
Thread t = new Thread(new ClientSocketThread(client));
t.setName(client.getInetAddress().getHostName() + ":" + counter++);
ExecutorService pool = Executors.newFixedThreadPool(MAX_CONNECTIONS);
pool.execute(t);

}
serverSocket.close();

} catch (IOException ex) {...}

}
```

Example 6

In the following example, the serve function receives an http request and an http response writer. It reads the entire request body.

(bad code)

Example Language: Go

```
func serve(w http.ResponseWriter, r *http.Request) {

var body []byte
if r.Body != nil {

if data, err := io.ReadAll(r.Body); err == nil {

body = data

}

}

}
```

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.

MaxBytesReader prevents clients from accidentally or maliciously sending a large request and wasting server resources. If possible, the code could be changed to tell ResponseWriter to close the connection after the limit has been reached.

(good code)

Example Language: Go

```
func serve(w http.ResponseWriter, r *http.Request) {

var body []byte
const MaxRespBodyLength = 1e6
if r.Body != nil {

r.Body = http.MaxBytesReader(w, r.Body, MaxRespBodyLength)
if data, err := io.ReadAll(r.Body); err == nil {

body = data

}

}

}
```

Selected Observed Examples

Note: this is a curated list of examples for users to understand the variety of ways in which this weakness can be introduced. It is not a complete list of all CVEs that are related to this CWE entry.

Reference	Description
CVE-2019-19911	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.
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.

Weakness Ordinalities

Ordinality	Description
Primary	(where the weakness exists independent of other weaknesses)
Resultant	(where the weakness is typically related to the presence of some other weaknesses)

Detection Methods

Method	Details

Automated Static Analysis	Automated static analysis typically has limited utility in recognizing resource exhaustion problems, except for program-independent system resources such as files, sockets, and processes. For system resources, automated static analysis may be able to detect circumstances in which resources are not released after they have expired. Automated analysis of configuration files may be able to detect settings that do not specify a maximum value. Automated static analysis tools will not be appropriate for detecting exhaustion of custom resources, such as an intended security policy in which a bulletin board user is only allowed to make a limited number of posts per day. Effectiveness: Limited
Automated Dynamic Analysis	Certain automated dynamic analysis techniques may be effective in spotting resource exhaustion problems, especially with resources such as processes, memory, and connections. The technique may involve generating a large number of requests to the product within a short time frame. Effectiveness: Moderate
Fuzzing	While fuzzing is typically geared toward finding low-level implementation bugs, it can inadvertently find resource exhaustion problems. This can occur when the fuzzer generates a large number of test cases but does not restart the targeted product in between test cases. If an individual test case produces a crash, but it does not do so reliably, then an inability to handle resource exhaustion may be the cause. Effectiveness: Opportunistic

Memberships

This MemberOf Relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	V View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.

MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	V View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an
MemberOf	V View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an
MemberOf	V View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an
MemberOf	C Category - a CWE entry that contains a set of other entries that share a common characteristic.
MemberOf	V View - a subset of CWE entries that provides a way of examining CWE content. The two main view structures are Slices (flat lists) an

Vulnerability Mapping Notes

Usage	**DISCOURAGED** (this CWE ID should not be used to map to real-world vulnerabilities)
Reason	Frequent Misuse
Rationale	CWE-400 is intended for incorrect behaviors in which the product is expected to track and restrict how many resources it consumes, but CWE-400 is often misused because it is conflated with the "technical impact" of vulnerabilities in which resource consumption occurs. It is sometimes used for low-information vulnerability reports. It is a level-1 Class (i.e., a child of a Pillar).

Comments	Closely analyze the specific mistake that is causing resource consumption, and perform a CWE mapping for that mistake. Consider children/descendants such as CWE-770: Allocation of Resources Without Limits or Throttling, CWE-771: Missing Reference to Active Allocated Resource, CWE-410: Insufficient Resource Pool, CWE-772: Missing Release of Resource after Effective Lifetime, CWE-834: Excessive Iteration, CWE-405: Asymmetric Resource Consumption (Amplification), and others.

Notes

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

"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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Resource exhaustion (file descriptor, disk space, sockets, ...)
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service
WASC	10		Denial of Service
WASC	41		XML Attribute Blowup
The CERT Oracle Secure Coding Standard for Java (2011)	SER12-J		Avoid memory and resource leaks during serialization
The CERT Oracle Secure Coding Standard for Java (2011)	MSC05-J		Do not exhaust heap space
Software Fault Patterns	SFP13		Unrestricted Consumption
ISA/IEC 62443	Part 3-3		Req SR 7.1
ISA/IEC 62443	Part 3-3		Req SR 7.2
ISA/IEC 62443	Part 4-1		Req SI-1
ISA/IEC 62443	Part 4-1		Req SVV-3
ISA/IEC 62443	Part 4-2		Req CR 7.1
ISA/IEC 62443	Part 4-2		Req CR 7.2

References

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 (CWE Draft 3, 2006-07-19)	CLASP	

Contributions

Contribution Date	Contributor	Organization
2023-01-24 (CWE 4.10, 2023-01-31)	"Mapping CWE to 62443" Sub-Working Group	CWE-CAPEC ICS/OT SIG
	Suggested mappings to ISA/IEC 62443.	
2023-04-25	"Mapping CWE to 62443" Sub-Working Group	CWE-CAPEC ICS/OT SIG
	Suggested mappings to ISA/IEC 62443.	
2025-02-25 (CWE 4.17, 2025-04-03)	Abhi Balakrishnan	
	Provided diagram to improve CWE usability.	

Modifications

Modification Date	Modifier	Organization
2025-12-11 (CWE 4.19, 2025-12-11)	CWE Content Team	MITRE
	updated Applicable_Platforms, Maintenance_Notes, Weakness_Ordinalities	
2025-09-09 (CWE 4.18, 2025-09-09)	CWE Content Team	MITRE
	updated Observed_Examples, References	
2025-04-03 (CWE 4.17, 2025-04-03)	CWE Content Team	MITRE
	updated Common_Consequences, Description, Diagram, Modes_of_Introduction, Other_Notes, Time_of_Introduction	
2024-11-19 (CWE 4.16, 2024-11-19)	CWE Content Team	MITRE
	updated Relationships	
2023-06-29	CWE Content Team	MITRE
	updated Mapping_Notes, Relationships	
2023-04-27	CWE Content Team	MITRE
	updated Demonstrative_Examples, Relationships, Taxonomy_Mappings	
2023-01-31	CWE Content Team	MITRE
	updated Common_Consequences, Description, Detection_Factors, Maintenance_Notes, Related_Attack_Patterns, Taxonomy_Mappings	
2022-10-13	CWE Content Team	MITRE
	updated Observed_Examples, Relationships	
2022-06-28	CWE Content Team	MITRE
	updated Observed_Examples, Relationships	
2022-04-28	CWE Content Team	MITRE
	updated Related_Attack_Patterns	
2020-08-20	CWE Content Team	MITRE
	updated Relationships	
2020-06-25	CWE Content Team	MITRE
	updated Description, Maintenance_Notes	
2020-02-24	CWE Content Team	MITRE

Submissions		
	updated Description, References, Related_Attack_Patterns, Relationships	
2019-09-19	CWE Content Team	MITRE
	updated Description, Relationships	
2019-06-20	CWE Content Team	MITRE
	updated Related_Attack_Patterns, Relationships	
2019-01-03	CWE Content Team	MITRE
	updated Alternate_Terms, Description, Name, Relationships, Taxonomy_Mappings, Theoretical_Notes	
2018-03-27	CWE Content Team	MITRE
	updated References, Type	
2017-11-08	CWE Content Team	MITRE
	updated Applicable_Platforms, Demonstrative_Examples, Likelihood_of_Exploit, Potential_Mitigations, References, Relationships	
2017-01-19	CWE Content Team	MITRE
	updated Relationships	
2015-12-07	CWE Content Team	MITRE
	updated Related_Attack_Patterns, Relationships	
2014-07-30	CWE Content Team	MITRE
	updated Relationships, Taxonomy_Mappings	
2013-07-17	CWE Content Team	MITRE
	updated Relationships	
2012-05-11	CWE Content Team	MITRE
	updated Demonstrative_Examples, Related_Attack_Patterns, Relationships, Taxonomy_Mappings	
2011-06-01	CWE Content Team	MITRE
	updated Common_Consequences, Relationships, Taxonomy_Mappings	
2010-09-27	CWE Content Team	MITRE
	updated Demonstrative_Examples	
2010-06-21	CWE Content Team	MITRE
	updated Description	
2010-04-05	CWE Content Team	MITRE
	updated Related_Attack_Patterns	
2010-02-16	CWE Content Team	MITRE
	updated Detection_Factors, Potential_Mitigations, References, Taxonomy_Mappings	
2009-12-28	CWE Content Team	MITRE
	updated Common_Consequences, Demonstrative_Examples, Detection_Factors, Likelihood_of_Exploit, Observed_Examples, Other_Notes, Potential_Mitigations, References	
2009-10-29	CWE Content Team	MITRE
	updated Relationships	
2009-07-27	CWE Content Team	MITRE

Submissions		
	updated Description, Relationships	
2009-05-27	CWE Content Team	MITRE
	updated Name, Relationships	
2009-01-12	CWE Content Team	MITRE
	updated Description	
2008-10-14	CWE Content Team	MITRE
	updated Description, Name, Relationships	
2008-09-08	CWE Content Team	MITRE
	updated Common_Consequences, Relationships, Other_Notes, Taxonomy_Mappings	
2008-08-15		Veracode
	Suggested OWASP Top Ten 2004 mapping	
2008-07-01	Eric Dalci	Cigital
	updated Time_of_Introduction	

Previous Entry Names

Change Date	Previous Entry Name
2008-10-14	Resource Exhaustion
2009-05-27	Uncontrolled Resource Consumption (aka 'Resource Exhaustion')
2019-01-03	Uncontrolled Resource Consumption ('Resource Exhaustion')
