## 7.CGY Concurrency Data Corruption [CGY]

Note: This vulnerability is placed in Clause 7 as an application vulnerability.

## 7.CGY.0 Terminology

Stateless Protocol: A communication or cooperation between threads where no state is preserved in the protocol itself (example HTTP or direct access to a shared resource). Since most interaction between threads require that state be preserved, the cooperating threads must use values of the resources(s) themselves or add additional communication exchanges to maintain state.

Stateless protocols require that the application provide explicit resource protection and locking mechanisms to guarantee the correct creation, view, access to, modification of, and destruction of the resource – i.e. the state needed for correct handling of the resource).

Process: A single execution of a program, or portion of an application. Processes often share a common processor, or network, or operating system or filing system or environment variables or other resources, but do not normally share a common memory space. Processes are usually started and stopped by an operating system and may or may not interact with other processes.

## 7.CGY.1 Description of Application Vulnerability

A resource that is directly visible from more than one process (at the same approximate time) and is not protected by access locks can be hijacked or used to corrupt, control or change the behaviour of other processes in the system. This corruption can lead to resource corruption, premature program termination, livelock, system corruption, or arbitrary execution of code.

Many vulnerabilities that are associated with concurrent access to files, shared memory or shared network resources fall under this vulnerability, including resources accessed via stateless protocols such as HTTP and remote file protocols.

## 7.CGY.2 Cross References

Burns A. and Wellings A., Language Vulnerabilities - Let’s not forget Concurrency, IRTAW 14, 2009.

CWE 642 External Control of Critical State Data

CWE 15 External Control of System or Configuration Setting

## 6.CGY.3 Mechanism of Failure

Any time that a shared resource is open to general inspection, the resource can be monitored by a foreign process to determine usage patterns, timing patterns, and access patterns to determine ways that a planned attack can succeed.[[1]](#footnote-2) Such monitoring could be, but are not limited to:

* Read resource values to obtain information of value to the applications
* Monitoring access time and access thread to determine when a resource can be accessed undetected by other threads (for example, Time-of-Check-Time-Of-Use attacks rely upon a determinable amount of time passage between the check on a resource and the use of the resource when the resource could be modified to bypass the check);
* Monitor a resource and modification patterns to help determine the protocols in use.
* Monitor access times and patterns to determine quiet times in the access to a resource that could be used to find successful attack vectors

This monitoring can then be used to construct a successful attack, usually in a later attack.

Any time that a resource is open to general update the resource can be changed by a foreign process to:

* Determine how such changes affect usage patterns, timing patterns, access patterns to determine ways that a planned attack can succeed.
* Determine how well application threads detect and respond to changed values

Any time that a shared resource is open to shared update by a thread, the resource can be changed in ways to further an attack once it is initiated. For example, in a well known attack, a process monitors a certain change to a known file and then immediately replaces a virus free file with an infected file to bypass virus checking software.

With careful planning, the above scenarios occurrence can result in the foreign process determining a weakness of the attacked process and an exploit consisting of anything up to and including arbitrary code execution.

### 7.CGY.5 Avoiding the Vulnerability or Mitigating its Effects

Software developers can avoid the vulnerability or mitigate its effects in the following ways.

* Place all shared resources in memory regions accessible to only one process at a time.
* Protect resources that must be visible with encryption or with checksums to detect unauthorized modifications.
* Protect access to shared resources using permissions, access control, or obfuscation.
* Have and enforce very clear rules with respect to permissions to change shared resources.
* Detect attempts to alter shared resources and take immediate action

1. Such monitoring is almost always possible by a process executing with system privilege, but even small slips in access controls and permissions lets such resources be seen from other (non system level) processes. Even the existence of the resource, its size, or its access dates/times and history (such as “last accessed time”) can give valuable information to an observer. [↑](#footnote-ref-2)