N0605

<<rename the 6.39 Vulnerability to "Memory Leaks and Heap Fragmentation" [XYL]>>

6.39.3 Mechanism of failure

As a process or system runs, any memory taken from dynamic memory and not returned or reclaimed (by the runtime system, the application, or a garbage collector) after it ceases to be used, may result in future memory allocation requests failing for lack of free space.

Alternatively, memory claimed and returned can cause the heap to fragment into progressively smaller blocks, which, with the usual allocators, will result in a higher memory consumption and steadily increasing search times for blocks of suitable size, until the system spends most of the CPU-time for searching the heap for suitable blocks.

Either condition can thus result in a memory exhaustion exception, progressively slower performance by the allocating application, program termination or a system crash.

If an attacker can determine the cause of an existing memory leak or can increase the allocation rate for blocks of different sizes, the attacker will be able to cause the application to leak or fragment quickly and therefore cause the application to crash or to perform within acceptable time limits. Denial-of-Service attacks can thus occur.

6.39.4 Applicable language characteristics

This vulnerability description is intended to be applicable to languages with the following characteristics:

- Languages reclaim memory under programmer control can exhibit heap fragmentation and memory leaks.
- Languages that that support mechanisms to dynamically allocate memory and employ garbage collection can exhibit memory leaks.

6.39.5 Avoiding the vulnerability or mitigating its effects

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

- Use garbage collectors that reclaim memory no longer accessible by the application. Some garbage collectors are part of the language while others are add-ons.
- In systems with garbage collectors, set all non-local pointers or references to null, when the designated data is no longer needed, since the data transitively reachable from such a pointer or reference will not be garbage-collected otherwise, effectively causing memory leaks..

Stephen Michell 2015-11-23 5:53 PM
Formatted: Normal

xxxxxx 2015-11-22 11:52 PM
Deleted:

ploedere 2015-11-22 4:49 PM
Deleted: eventually result in an inability to allocate the necessary size storage.

ploedere 2015-11-22 4:49 PM
Deleted:

ploedere 2015-11-22 4:50 PM
Deleted: will

ploedere 2015-11-22 4:51 PM
Deleted: and

ploedere 2015-11-22 4:57 PM
Deleted: may

ploedere 2015-11-22 4:58 PM
Formatted: Font:+Theme Body

ploedere 2015-11-22 4:58 PM
Formatted: Font:+Theme Body

ploedere 2015-11-22 5:02 PM
Deleted: that support mechanisms to dynamically allocate memory and

ploedere 2015-11-22 5:02 PM
Formatted: List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm

ploedere 2015-11-22 5:03 PM
Deleted:

- In systems without garbage collectors, cause deallocation of the data before the last pointer or reference to the data is lost.
- Allocate and free memory at the same level of abstraction, and ideally in the same code module. Allocating and freeing memory in different modules and levels of abstraction may make it difficult for developers to match requests to free storage with the appropriate storage allocation request. This may cause confusion regarding when and if a block of memory has been allocated or freed, leading to memory leaks.
- Use Storage pools when available in combination with strong typing. Storage pools are a specialized memory mechanism where all of the memory associated with a class of objects is allocated from a specific bounded region such that storage exhaustion in one pool does not affect the code operating on other memory.
- Use storage pools of equally-sized blocks to avoid fragmentation within each storage pool. If necessary, provide application-specific (de-)allocators to achieve this functionality.
- Avoid the use of dynamically allocated storage entirely, or allocate only during system initialization and never allocate once the main execution commences, particularly in safety-critical systems and long running systems.
- Use static analysis, which can sometimes detect when allocated storage is no longer used and has not been freed.

ploedere 2015-11-22 5:04 PM
Deleted:

ploedere 2015-11-22 5:05 PM
Formatted: Indent: Left: 1.25 cm

ploedere 2015-11-22 5:07 PM
Deleted: To avoid these situations,

ploedere 2015-11-22 5:08 PM
Deleted: