메모리 에러 코드

김성익(noerror@hitel.net)
2004.03.18

개요

- 런타임상 문제들
- 메모리 문제로 인한 버그
  발견의 어려움

- DevPartner BoundChecker 는 ?

- 메모리사용 / 포인터 사용
  메모리 leak
  버그의 위험성을 가지는 코드

Assigning Pointer Out of Range

- 할당된 메모리를 넘어선 영역을 지시하는 포인터

```
#define ELEMENT_LEN 20

void TestePointerRangeErrorVar(void)
{
    int rgiElements[ELEMENT_LEN];
    int *pint = rgiElements + ELEMENT_LEN - 1;

    ++pint; //legal
    ++pint; //illegal
}
```

- 마지막 영역의 계산상 사소한 실수
- 사용하지 않더라도 위험

```
    void* ptr = malloc(4);
    void *ptr2 = (char*)ptr + 5;

    // Two errors are generated here:
    // The right side evaluation will generate a 'pointer arithmetic range error'.
    // The assignment to ptr2 will generate a 'pointer assignment range error'.

    void *ptr3 = myFun(ptr2);

    // - this line gives two errors:
    // using ptr2 will generate a 'pointer argument range error'.
    // The assignment to ptr3 will generate a 'pointer assignment range error'.
```

Pointer Out of Block Range

- 포인터가 사용 가능한 메모리 블록 이외를 지시하는 경우

```
#include <string.h>
#include <mbstring.h>
#include <windows.h>

int main(int argc, char* argv[])
{
    byte     buffer[20];
    byte*    pStart;
    byte*    pTemp;

    // This function is intended to decrement a pointer
    // within a block, based on knowing the starting address
    // of the block.
    //
    // pTemp is out of range of the buffer starting at pStart

    pStart    = buffer;
    pTemp     = pStart + 40;
    pTemp     = _mbsdec(pStart, pTemp);

    return 0;
}
```

Reading/Writing Overflows Memory

- 사용 가능한 영역을 넘어서 읽는 경우

```
char array[10];
int *ptr = (int*)( array + 8);   // this pointer is valid, as it points within the array
int n = *ptr;                    // error, because we're within 4 bytes of the end
```

- 사용 가능한 영역을 넘어서 쓰는 경우

```
...
char z[10];
strcpy (z, "A simple test");
...
```

Unallocated Pointer

- 메모리 할당되지 않는, 유효하지 않은 포인터

```
int main(int argc, char* argv[])
{
    char * ptr = (char *) 0x0BADFOOD;
```

Reading Uninitialized Memory

- 초기화되지 않은 메모리를 읽는 경우

```
struct square box;
int area;
box.width = 5;
area = box.width*box.height; // Height is not initialized
printf("area = %d₩n", area);
...
```

Bad Write Pointer

- Write가 불가능한(무의미한) 메모리에 write 를 시도하는 경우

```
#include <string.h>

int main(int argc, char* argv[])
{
    char    string[] = "Test";
    // Read only destination pointer.
    strcpy("Read only literal string", string);
    return 0;
}
```

Static Memory Overrun

- 정적 메모리의 영역을 벗어나서 억세스 하는 경우

```
void function_1 ( void )
{
    static char array[10];
    strcpy ( array, "This will overwrite ");
}
```

Dynamic Memory Overrun

- 할당된 메모리를 벗어나 접근하는 경우

```
char*a = (char*)malloc (10);
memset (a, 0, 12);
```

- 소스 문자열의 '₩0'이 없는 경우
  소스 문자열이 너무 긴 경우
  길이 인자가 잘못되는 경우

Buffer or Parameter Is Not Null Terminated

- 문자열 끝에 '₩0'이 없어서 문자열의 길이가 제대로 파악이 안 되는 경우

```
int main(int argc, char* argv[])
{
    char    * string;
    char    * buffer;

    string = (char *) malloc(10);
    buffer = (char *) LocalAlloc(LPTR, 100);

    strncpy(string, "123456789A", 10);

    // Error: Source String is not terminated
    strcpy(buffer, string);

    return 0;
}
```

Subscript Overrun: Array Subscript Value Exceeds Array Size

- 배열의 벗어난 인덱스로 접근하는 경우 (실제 메모리는 유효해서 safe하더라도 버그 유발 가능성)

```
char buffer[10][20];
int n = 3;
int m = 25;
size_t diff = &buffer[n][m] - buffer[0];
```

Dangling Allocation Reference

- 메모리 해제된 메모리를 가리키는 포인터사용

```
char *f = (char *)malloc(10);
char g[10];
free(f);
if (f > g)
    f = g;
```

- 버그를 유발하는 코드가 만들어 질 수 있다.

- 모두 사용 후 해제하거나, 메모리 해제된 메모리 포인터는 사용하지 않는다.

Bad Pointer

- 유효하지 않는 포인터

- Unload된 DLLS에서 사용하던 메모리
- 해제된 리소스의 메모리 맵
- 더 이상 유효하지 않는 스택
- 메모리 해제된 힙메모리
- 사용할 수 없는 메모리 영역

Unrelated Pointer Comparison

- 서로 연관성이 없는 포인터간의 비교 연산

```
void PointerCompareUnrelated()
{
    HANDLE     hHeap1;
    HANDLE     hHeap2;
    LPVOID     pBuffer1; // Buffer in heap 1
    LPVOID     pBuffer2; // Buffer in heap 2
    char       StackBuffer[100];

    hHeap1 = HeapCreate(0, 4096, 8192);
    hHeap2 = HeapCreate(0, 4096, 8192);

    pBuffer1 = HeapAlloc(hHeap1, HEAP_ZERO_MEMORY, 100);
    pBuffer2 = HeapAlloc(hHeap2, HEAP_ZERO_MEMORY, 100);

    // The following two comparison should cause
    // unrelated pointer errors

    if (pBuffer1 < pBuffer2)
        printf("Comparing the address of two different heap blocks is undefined.₩n");

    if (pBuffer1 < &StackBuffer)
        printf("Comparing heap and stack addresses is undefined.₩n");

    HeapFree(hHeap1, 0, pBuffer1); HeapDestroy(hHeap1);
    HeapFree(hHeap2, 0, pBuffer2); HeapDestroy(hHeap2);
}
```

Memory Leak

- 메모리를 할당하고 해제하지 않는 경우

```
void function_test()
{
    int *intptr;
    intptr = new int [1000];
}
```

- 사용한 메모리는 반드시 해제한다

Interface Leak

- 생성한 리소스를 release하지 않고 다시 할당하는 경우 최종적으로 ref 카운트가 0가 안 되서 해제가 안 되는 경우

```
LPUNKNOWN lpUnk ;
HRESULT hr = CoCreateInstance (clsidIFace, NULL, CLSCTX_INPROC_SERVER, IID_IUnknown, (LPVOID*)&lpUnk) ;
```

Leak Due to Leaked Block

- 메모리 leak 난 블록으로 인해서 leak 이 유발

```
struct mystruct
{
    void* ptr; // Pointer to a block of memory
    int i;
    mystruct() {ptr = malloc (19); }
    ~mystruct() { free(ptr); }
};

void LeakDuetoLeak()
{
    // Allocate a structure. Note that this structure
    // will automatically allocate a block of memory and
    // assign it to "ptr" within mystruct.

    mystruct* pBlock = new mystruct;
}
```

Memory Leaked Due to Unwind

- 예외로 리턴 되는 경우 할당된 메모리가 해제 안 되는 경우

```
// leak_unwind.cpp : Defines the entry point for the console application.

#include "stdio.h"
#include "stdlib.h"

void GenerateException(void)
{
    char *Buffer = (char *) malloc(20);
    int *BadPointer = NULL;
    *BadPointer = 12345; // Generate an exception
    return;
}

void LeakDueToExceptionTest(void) // try ... catch block
{
    try
        {
            GenerateException();
        }

    catch (...)
        {
            printf("Caught the exception...#n");
        }

    return;
}
```

Memory Leaked Due to Free

- 객체 소멸 시 사용한 맴버 포인터의 메모리를 해제하지 않을 경우

```
...
typedef struct ptrblock
{
    char *ptr;
}   AZ;
...

AZ *p;

p = (AZ *)malloc(sizeof(*p));
p->ptr = (char*)malloc(10);
free(p);
...
```

Memory Leaked Due to Reassignment

- 사용중인 포인터에 다른 주소를 넣어 이전 메모리 leak 이 생기는 경우

```
...
char *c, d[10];
c = (char *)malloc(10);
c = d;
...
```

Memory Leaked Leaving Scope

- 함수 내에서 메모리를 할당하고 외부로 전달하지도 않고, 해제하지도 않는 경우

```
void getmem()
{
    char *d;
    d = (char*)malloc(10);
    return;
}

int main(int argc, char* argv[])
{
    getmem();
    return 0;
}
```

Function Pointer Is Not a Function

- 함수 포인터의 위치가 실제 함수의 주소가 아니다

```
union {
    int *iptr;
    int (*fptr) ();
} u;
...
int i;
u.iptr = &i;
u.fptr();
...
```

- *(Bad Indirect Call)*

Returning Pointer to Local Variable

- 로컬 변수를 리턴 하는 경우

```
char *MakeFullName(char *FirstName)
{
    char array[80];

    strcpy(array, FirstName);
    strcat(array, "Smith");
    // Return pointer to local variable
    return array;
}
```

- static 변수로 만들거나, 동적할당 하거나, 전역 변수로 만든다

Resource Leak on Exit

- 리소스를 사용 후 해제하지 않는 경우

```
#include <windows.h>

class CLoadLibrary
{
public:
    CLoadLibrary()
    {
        m_hWinsockDll = ::LoadLibraryA("mpr.dll");
    }
    ~CLoadLibrary(){}

private:
    HMODULE m_hLoadedDll;
};

int main(int argc, char* argv[])
{
    CLoadLibrary    LoadedDll;

    return 0;
}
```

- 해제한다

Pointer Passed When a Handle Was Expected

- 잘못된 함수 사용으로 메모리가 해제되거나, 재할당되어야 하는 시도가 실패했을 때, 별도의 처리가 없는 경우

```
void AccessHandlePtr(void)
{
    LPVOID hHandleA = GlobalAlloc(GMEM_MOVEABLE, 47);

    //a block with both handle and ptr.

    void* pPointer = GlobalLock(hHandleA);

    // Error: Invalid handle in GlobalRealloc, a pointer
    //        to a locked memory block can not be used as
    //        handle to reallocate a new block.

    void* pNewPointer = GlobalReAlloc(pPointer, 59, GMEM_MOVEABLE);
}
```

- GlobalAlloc 관련 함수들처럼 포인터 대신 핸들을 사용하는 함수는 함수 레퍼런스를 충분히 숙지

Pointer Not at the Beginning of the Allocated Block

- 할당된 메모리 포인터의 위치를 입력으로 받는 함수에 증감된 위치의 포인터를 넘기는 경우

```
#include <windows.h>

int main(int argc, char* argv[])
{
    HANDLE     hHeap         = GetProcessHeap();
    void *     pPointer1     = HeapAlloc(hHeap, 0, 42);
    size_t     nBlockSize1   = HeapSize(hHeap, 0, pPointer1);
    void*      pPointer2     = (char*) pPointer1 + 6;
    DWORD      nSize;

    // Error: pPointer2 does not point to the start of the heap block

    nSize = HeapSize(hHeap, 0, pPointer2);

    HeapFree(hHeap, 0, pPointer1);

    return 0;
}
```

Memory Allocation Conflict: Function Mismatch

- 메모리 할당자와 해제자가 다른 경우

```
...
char *a;
a = new char;
free(a);
return 0;
...
```

Memory Allocation Conflict: Module Conflict

- 다른 모듈에서 할당한 메모리를 해제하는 경우

결론

- Overflowing a heap buffer is extremely dangerous
- 항상 예측 가능해야 한다
  모든 경우를 대비한다
- 집중력이 떨어진 상태에서 코딩하지 않는다

생략한 이슈들

- Destroying Process Heap
- Freed Handle Is Still Locked
- Function Refers to a Zero Length Block
- Handle Is Already Unlocked
- Inefficient Use of GlobalAlloc/LocalAlloc
- Nonzero Lock Count

Handle Passed Instead of a Pointer to Memory

- 메모리 함수의 결과가 메모리 포인터가 아니라 핸들인 경우

```
#include "windows.h"
int main(int argc, char * argv [])
{
	// Example, calling LocalHandle() with a handle, rather
	// than with a (locked) pointer. The function should
	// only be used on pointers, and only pointers to
	// moveable blocks.
	HLOCAL handle = LocalAlloc (GMEM_MOVEABLE, 42);

	//Error, should be passing in a pointer
	HLOCAL h2 = LocalHandle (handle);

	LocalFree (handle);

	// Example, calling GlobalHandle() with a handle, rather
	// than with a (locked) pointer. The function should
	// only be used on pointers, and only pointers to
	// moveable blocks.
	HLOCAL hGlobal = GlobalAlloc (GMEM_MOVEABLE, 42);

	//Error, should be passing in a pointer
	HLOCAL hNewHandle = GlobalHandle (hGlobal);

	GlobalFree (hGlobal);
	return 0;
}
```

Pointer References Unlocked Memory Block

- Lock 을 해야 유효한 포인터를 Lock 없이 사용

```
...
HANDLE hMem;
LPVOID lpPtr;
hMem = GlobalAlloc(GHND, 0x100);    //Allocate moveable memory
lpPtr = GlobalLock(hMem);           //Get pointer to memory
GlobalUnlock(hMem);                 //Unlock pointer
memset(lpPtr,0,0x100);              //Use pointer and BC complains
GlobalFree(hMem);
...
```

Bad Heap Handle

- 적절하지 않은 핸들로 heap함수를 호출하는 경우

```
#include "windows.h"
int main(int argc, char* argv[])
{
    HANDLE  hHeap1;
    HANDLE  hHeap2;
    void * pBlock;
    BOOL  status;
    hHeap1 = HeapCreate(0, 8192, 16384);
    hHeap2 = HeapCreate(0, 8192, 16384);
    pBlock = HeapAlloc(hHeap1, HEAP_ZERO_MEMORY, 64);
    // Error: Should be hHeap1
    status = HeapFree(hHeap2, 0, pBlock);
    return 0;
}
```

Commit Exceeds Reserve

- VirtualAlloc 함수에서 예약된 메모리보다 큰 메모리를 할당하는 경우

```
DWORD Size = 2048;
LPVOID Memory = VirtualAlloc( NULL,
                              Size,
                              MEM_RESERVE,      // Allocation type
                              PAGE_READWRITE    // Page Protection
                              );

// now commit it but for a much bigger size

VirtualAlloc( Memory,           // desired address - from the previous reserve
              Size * 10,
              MEM_COMMIT,       // Allocation type
              PAGE_READWRITE    // Page Protection
              );
```