|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **2008-11-21**   |  | | --- | | [Do not reture memory malloc in unmanage code to manage code.](http://teams.collaboration.agilent.com/sites/bid/vee/blog/Lists/Posts/Post.aspx?ID=34) |   Haitao's email:  **VEE will crash** in Vista if return a memory malloc in unmanage code to manage code,  We have some functions in C:\VEE\main\Src\Veeman\VeerunInterop\VeerunNativeMethods.cs file like,    [DllImport(VeeRunDllFileName, CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]  public static extern **string** slnGetRootCarrierFilename();  ……  [DllImport(VeeRunDllFileName, CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]  public static extern **string** rootCtxGetPopupTitle();  ……    Use slnGetRootCarrierFilename function as sample  The implement of this function in the C:\vee\main\Src\IPCore\RootCarrier.m file    \_\_declspec(dllexport) char\* slnGetRootCarrierFilename()  {      return [g\_rootCarrier **currentFileName**];  }  The **currentFileName variable is malloc** in unmanage Heap, so it will crash after the slnGetRootCarrierFilename function be called from manage code.    we will have a short meeting, QiLuo will show how to write the correct code in such situation.     Please refence <http://forums.microsoft.com/msdn/ShowPost.aspx?postid=1677959&siteid=1>    Luo Qi's solution:  Yeah, the cause is that the interop marshaler always attempts to free memory allocated by unmanaged code, except memory referenced by an IntPtr type.  CLR doesn’t automatically free IntPtr type pointers.    We get 3 ways out of this situation.   1. In Haitao’s post, use CoTaskMemAlloc instead of malloc.   For example,  void\* allocAndReturnStr()          {              char\* str = (char\*)CoTaskMemAlloc(100);              strcpy(temp, "test!");              return (void\*)str;          }            Then at your managed call, you should,          [DllImport(unmanaged.dll, CallingConvention = CallingConvention.StdCall, CharSet = CharSet.Ansi)]          public static extern IntPtr GetStrFromAlloc();            …          IntPtr ptr = GetStrFromAlloc()          String str = Marshal. PtrToStringAnsi();   Marshal.FreeCoTaskMem(ptr);    It’s applied to the case when you have to allocate memory, and also be available for you wanna pass strings array from unmanaged to managed.     1. Pass a buffer into unmanaged function, get it filled, then get returned value. The flaw is we won’t know the size of buffer in advance and will have to rely upon the unmanaged function to provide its size, exactly like what Sue did.    int len = VeerunNativeMethods.propDescriptionStrlen(pVeeProp);     // decide buffer size           StringBuilder builder = new StringBuilder(len);           VeerunNativeMethods.propCopyDescription(pVeeProp, builder, len);   // get buffer filled    String desc = builder.ToString();     1. You can avoid automatic memory freeing in situations where memory should never be freed, such as when using the GetCommandLine function from Kernel32.dll, which returns a pointer to kernel memory.      IntPtr cmdLineStr = Win32NativeMethods.GetCommandLine();          String commandLine = Marshal.PtrToStringAuto(cmdLineStr);                  In addition, it’s convenient to apply item3 in many VEE cases, for example,                [DllImport(VeeRunDllFileName, CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]          public static extern IntPtr slnGetRootCarrierFilename();                  Marshal.PtrToStringAnsi(VeerunNativeMethods.slnGetRootCarrierFilename());            Since VEE takes care of how and when to free memory, we don’t worry about that.    JiaDong's email:  Might I simplify the rule created by Czar Luo?      1. **If the return value is a new memory:**   **In unmanaged side:**   * 1. You should realloc a new memory by CoTaskMemAlloc;   2. Strcpy and free   3. return memory alloced by CoTaskMemAlloc;   For example:  \_\_declspec(dllexport) char\* compiledFunctionCallingName(id aCompiledUserfunction)  {     char\* name = [aCompiledUserfunction nameString]; // new memory should be freed     char\* retval = CoTaskMemAlloc(sc\_strlen(name));     sc\_strcpy(retval, name);     sc\_free(name);     return retval;  }    **In managed side:**  Declare you P/Invoke method like:  public static extern string compiledUserfunctionCallingName(IntPtr aCompiledFunction);    Note: if the return value is string, CLR will be responsible for free the memory alloced by CoTaskMemAlloc. | |