Why are structure sizes checked strictly?

devblogs.microsoft.com/oldnewthing/20031212-00

December 12, 2003

Raymond Chen

You may have noticed that Windows as a general rule checks structure sizes strictly. For example, consider the MENUITEMINFO structure:

```
typedef struct tagMENUITEMINFO {
  UINT    cbSize;
  UINT    fMask;
  UINT    fType;
  UINT    fState;
  UINT    wID;
  HMENU   hSubMenu;
  HBITMAP hbmpChecked;
  HBITMAP hbmpUnchecked;
  ULONG_PTR dwItemData;
  LPTSTR  dwTypeData;
  UINT    cch;
#if(WINVER >= 0x0500)
  HBITMAP hbmpItem; // available only on Windows 2000 and higher
#endif
} MENUITEMINFO, *LPMENUITEMINFO;
```

Notice that the size of this structure changes depending on whether WINVER >= 0x0500 (*i.e.*, whether you are targetting Windows 2000 or higher). If you take the Windows 2000 version of this structure and pass it to Windows NT 4, the call will fail since the sizes don't match.

"But the old version of the operating system should accept any size that is greater than or equal to the size it expects. A larger value means that the structure came from a newer version of the program, and it should just ignore the parts it doesn't understand."

We tried that. It didn't work.

Consider the following imaginary sized structure and a function that consumes it. This will be used as the guinea pig for the discussion to follow:

```
typedef struct tagIMAGINARY {
  UINT cbSize;
  BOOL fDance;
  BOOL fSing;
#if IMAGINARY_VERSION >= 2
  // v2 added new features
  IServiceProvider *psp; // where to get more info
#endif
} IMAGINARY;
// perform the actions you specify
STDAPI DoImaginaryThing(const IMAGINARY *pimg);
// query what things are currently happening
STDAPI GetImaginaryThing(IMAGINARY *pimg);
```

First, we found lots of programs which simply forgot to initialize the `cbSize` member altogether.

```
IMAGINARY img;
img.fDance = TRUE;
img.fSing = FALSE;
DoImaginaryThing(&img);
```

So they got stack garbage as their size. The stack garbage happened to be a large number, so it passed the "greater than or equal to the expected `cbSize`" test and the code worked. Then the next version of the header file expanded the structure, using the `cbSize` to detect whether the caller is using the old or new style. Now, the stack garbage is still greater than or equal to the new `cbSize`, so version 2 of `DoImaginaryThing` says, "Oh cool, this is somebody who wants to provide additional information via the `IServiceProvider` field." Except of course that it's stack garbage, so calling the `IServiceProvider::QueryService` method crashes.

Now consider this related scenario:

```
IMAGINARY img;
GetImaginaryThing(&img);
```

The next version of the header file expanded the structure, and the stack garbage happened to be a large number, so it passed the "greater than or equal to the expected `cbSize`" test, so it returned not just the `fDance` and `fSing` flags, but also returned an `psp`. Oops, but the caller was compiled with v1, so its structure doesn't have a `psp` member. The `psp` gets written past the end of the structure, corrupting whatever came after it in memory. Ah, so now we have one of those dreaded **buffer overflow** bugs.

Even if you were lucky and the memory that came afterwards was safe to corrupt, you still have a bug: By the rules of COM reference counts, when a function returns an interface pointer, it is the caller's responsibility to release the pointer when no longer needed. But the

v1 caller doesn't know about this `psp` member, so it certainly doesn't know that it needs to be `psp->Release()` d. So now, in addition to memory corruption (as if that wasn't bad enough), you also have a memory leak.

Wait, I'm not done yet. Now let's see what happens when a program written in the future runs on an older system.

Suppose somebody is writing their program intending it to be run on v2. They set the `cbSize` to the larger v2 structure size and set the `psp` member to a service provider that performs security checks before allowing any singing or dancing to take place. (*E.g.*, makes sure everybody paid the entrance fee.) Now somebody takes this program and runs it on v1. The new v2 structure size passes the "greater than or equal to the v1 structure size" test, so v1 will accept the structure and Do the ImaginaryThing. Except that v1 didn't support the `psp` field, so your service provider never gets called and your security module is bypassed. Now everybody is coming into your club without paying.

Now, you might say, "Well those are just buggy programs. They deserve to lose." If you stand by that logic, then prepare to take the heat when you read magazine articles like "Microsoft intentionally designed <Product X> to be incompatible with <software from a major competitor>. Where is the Justice Department when you need them?"

Raymond Chen

Follow