A function pointer cast is a bug waiting to happen

devblogs.microsoft.com/oldnewthing/20110506-00

May 6, 2011

Raymond Chen

A customer reported an application compatibility bug in Windows.

We have some code that manages a Win32 button control. During button creation, we subclass the window by calling `SetWindowSubclass`. On the previous version of Windows, the subclass procedure receives the following messages, in order:

- `WM_WINDOWPOSCHANGING`
- `WM_NCCALCSIZE`
- `WM_WINDOWPOSCHANGED`

We do not handle any of these messages and pass them through to `DefSubclassProc`. On the latest version of Windows, we get only the first two messages, and `comctl32` crashes while it's handling the third message before it gets a chance to call us. It looks like it's reading from invalid memory.

The callback function goes like this:

```
LRESULT ButtonSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam,
    UINT_PTR idSubclass,
    DWORD_PTR dwRefData);
```

We install the subclass function like this:

```
SetWindowSubclass(
    hwndButton,
    reinterpret_cast<SUBCLASSPROC>(ButtonSubclassProc),
    id,
    reinterpret_cast<DWORD_PTR>(pInfo));
```

We found that if we changed the callback function declaration to

```
LRESULT CALLBACK ButtonSubclassProc(
    HWND hwnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam,
    UINT_PTR idSubclass,
    DWORD_PTR dwRefData);
```

and install the subclass function like this:

```
SetWindowSubclass(
    hwndButton,
    ButtonSubclassProc,
    id,
    reinterpret_cast<DWORD_PTR>(pInfo));
```

> then the problem goes away. It looks like the new version of Windows introduced a compatibility bug; the old code works fine on all previous versions of Windows.

Actually, you had the problem on earlier versions of Windows, too. You were just lucky that the bug wasn't a crashing bug. But now it is.

This is a classic case of mismatching the calling convention. The `SUBCLASSPROC` function is declared as requiring the `CALLBACK` calling convention (which on x86 maps to `__stdcall`), but the code declared it without any calling convention at all, and the ambient calling convention was `__cdecl`. When they went to compile the code, they got a compiler error that said something like this:

```
error C2664: 'SetWindowSubclass' : cannot convert parameter 2 from 'LRESULT
(__cdecl *)(HWND,UINT,WPARAM,LPARAM,UINT_PTR,DWORD_PTR)' to 'SUBCLASSPROC'
```

"Since the compiler was unable to convert the parameter, let's give it some help and stick a cast in front. There, that shut up the compiler. Those compiler guys are so stupid. They can't even figure out how to convert one function pointer to another. I bet they need help wiping their butts when they go to the bathroom."

And there you go, you inserted a cast to shut up the compiler and masked a bug instead of fixing it.

The only thing you can do with a function pointer after casting it is to cast it back to its original type.[1] If you try to use it as the cast type, you will crash. Maybe not today, maybe not tomorrow, but someday.

In this case, the calling convention mismatch resulted in the stack being mismatched when the function returns. It looks like earlier versions of Windows managed to hobble along long enough before things got resynchronized (by an `EBP` frame restoration, most likely) so the damage didn't spread very far. But the new version of Windows, possibly one compiled with more aggressive optimizations, ran into trouble before things resynchronized, and thus occurred the crash.

The compiler was yelling at you for a reason.

It so happened that the Windows application compatibility team had already encountered this problem in their test labs, and a shim had already been developed to auto-correct this mistake. (Actually, the shim also corrects another mistake they hadn't noticed yet: They forgot to call `RemoveWindowSubclass` when they were done.)

[1]I refer here to pointers to static functions. Pointers to member functions are entirely different animals.

Raymond Chen

Follow