When programs grovel into undocumented structures…

devblogs.microsoft.com/oldnewthing/20031223-00

December 23, 2003

Raymond Chen

Three examples off the top of my head of the consequences of grovelling into and relying on undocumented structures.

Defragmenting things that can't be defragmented
In Windows 2000, there are several categories of things that cannot be defragmented. Directories, exclusively-opened files, the MFT, the pagefile… That didn't stop a certain software company from doing it anyway in their defragmenting software. They went into kernel mode, reverse-engineered NTFS's data structures, and modified them on the fly. Yee-haw cowboy! And then when the NTFS folks added support for defragmenting the MFT to Windows XP, these programs went in, modified NTFS's data structures (which changed in the meanwhile), and corrupted your disk. Of course there was no mention of this illicit behavior in the documentation. So when the background defragmenter corrupted their disks, Microsoft got the blame.

Parsing the Explorer view data structures
A certain software company decided that they wanted to alter the behavior of the Explorer window from a shell extension. Since there is no way to do this (a shell extension is not supposed to mess with the view; the view belongs to the user), they decided to do it themselves anyway.
From the shell extension, they used an undocumented window message to get a pointer to one of the internal Explorer structures. Then they walked the structure until they found something they recognized. Then they knew, "The thing immediately after the thing that I recognize is the thing that I want."

Well, the thing that they recognize and the thing that they want happened to be base classes of a multiply-derived class. If you have a class with multiple base classes, there is no guarantee from the compiler which order the base classes will be ordered. It so happened that they appeared in the order X,Y,Z in all the versions of Windows this software company tested against.

Except Windows 2000.

In Windows 2000, the compiler decided that the order should be X,Z,Y. So now they grovelled in, saw the “X” and said “Aha, the next thing must be a Y” but instead they got a Z. And then they crashed your system some time later.

So I had to create a “fake X,Y” so when the program went looking for X (so it could grab Y), it found the fake one first.

This took the good part of a week to figure out.

Reaching up the stack

A certain software company decided that it was too hard to take the coordinates of the NM_DBLCLK notification and hit-test it against the treeview to see what was double-clicked. So instead, they take the address of the NMHDR structure passed to the notification, **add 60 to it**, and dereference a DWORD at that address. If it’s zero, they do one thing, and if it’s nonzero they do some other thing.
It so happens that the NMHDR is allocated on the stack, so this program is reaching up into the stack and grabbing the value of some local variable (which happens to be two frames up the stack!) and using it to control their logic.

For Windows 2000, we upgraded the compiler to a version which did a better job of reordering and re-using local variables, and now the program couldn’t find the local variable it wanted and stopped working.

I got tagged to investigate and fix this. I had to create a special NMHDR structure that “looked like” the stack the program wanted to see and pass that special “fake stack”.

I think this one took me two days to figure out.

I hope you understand why I tend to go ballistic when people recommend relying on undocumented behavior. These weren’t hobbyists in their garage seeing what they could do. These were major companies writing commercial software. When you upgrade to the next version of Windows and you experience (a) disk corruption, (b) sporadic Explore crashes, or (c) sporadic loss of functionality in your favorite program, do you blame the program or do you blame Windows?

If you say, “I blame the program,” the first problem is of course figuring out which program. In cases (a) and (b), the offending program isn’t obvious.

Raymond Chen

Follow