Why does GetModuleInfo fail to produce an entry point for executables?

devblogs.microsoft.com/oldnewthing/20211014-00

October 14, 2021

Raymond Chen

A customer wanted to find the entry point for some executables. They tried this two-step plan:

- Load the executable via `LoadLibrary`. Like, a real `LoadLibrary`, none of this "as datafile" stuff.
- Call `GetModuleInformation` and look at the `MODULEINFO.EntryPoint`

The customer found that the result is always `NULL`. What's going on?

First of all, it's not clear what the purpose of the exercise is. You can't call the entry point, since it's not the entry point for *this* process. And the entry point obtained from the current process may not match the entry point when the process is actually run as a process.

Loading an executable as a library is a pretty dodgy operation. The module can be used to load resources, but you can't do much else with it. And if all you are after are the resources, you may as well just load it as a datafile.

What you can do is manually walk the Portable Executable header (either by seeking and reading or by using a memory-mapped file) to get the `AddressOfEntryPoint`, which is a relative virtual address. You can then add that to the base address of the module (when it eventually loads) to locate the entry point.

Okay, fine. But why does `GetModuleInformation` return `NULL` for executables?

Internally, `GetModuleInformation` walks the loader's internal list of loaded modules and returns the entry point from the loader entry for said module. If you load an executable via `LoadLibrary`, the entry point entry in the loader data structure will not be set.

The loader doesn't record the entry point for executable modules because executable modules don't receive `DLL_PROCESS_ATTACH` notifications. The entry point for an executable module is the process entry point, not the `DllMain` function. (The loader also doesn't record the entry point for CLR DLLs since that entry point is a dummy function that crashes on purpose.)

Basically, the deal is that `GetModuleInformation` is in `PSAPI.DLL`, and `PSAPI.DLL` goes "I'm in ur process steeling ur data". Not only does the loader have no use for entry points of EXEs and CLR DLLs, it explicitly *must not* call those entry points, so it sets the entry point to null to mean "Don't call this guy." This is a private thing inside the loader. `PSAPI.DLL` goes in and steals it and says "Hey check out all this good stuff I stole!" You're basically getting grey market data.

Raymond Chen

Follow