Why does the Windows Portable Executable (PE) format have separate tables for import names and import addresses?, part 1

devblogs.microsoft.com/oldnewthing/20231129-00

November 29, 2023

Raymond Chen

In the Windows Portable Executable (PE) format, the image import descriptor table describes the functions imported from a specific target DLL.

```
struct IMAGE_IMPORT_DESCRIPTOR {
    DWORD   OriginalFirstThunk;
    DWORD   TimeDateStamp;
    DWORD   ForwarderChain;
    DWORD   Name;
    DWORD   FirstThunk;
};
```

The `OriginalFirstThunk` points to an array of pointer-sized `IMAGE_THUNK_DATA` structures which describe the functions being imported. The `FirstThunk` points to an array of pointers, whose initial values are a copy of the values pointed to by `OriginalFirstThunk`. When the DLL is loaded, those initial values in the `FirstThunk` table are replaced by the actual function pointers determined at runtime.

But why are there two copies of the table? The two tables are never needed at the same time, so why not reuse the memory? When the DLL is initially loaded, the entries describe the functions being imported, and after the function addresses are located, they could be written back into the same table.

The answer is DLL binding.

As a load-time optimization, you can *bind* your DLL to its targets. If the target DLL has `0x20304000` as its preferred base address, then if the DLL gets loaded at that preferred base address, you know what all the function addresses are going to be, and *binding* records those precalculated function addresses into the `FirstThunk` table. After binding is performed, the `FirstThunk` table now holds the precalculated function addresses and is not a copy of the `OriginalFirstThunk` table. The module timestamp of the DLL that was used to calculate the bindings is recorded in the image import directory.[1]

When the DLL is loaded, the loader checks whether the module timestamp recorded in the image import descriptor matches the timestamp of the actual module found at runtime. If so, then it just uses the precalculated values in the `FirstThunk` table. And if not, then the loader uses the `OriginalFirstThunk` table to look up the functions at runtime.

Therefore, you can't combine the `OriginalFirstThunk` and `FirstThunk` tables: If the precalculated values in the `FirstThunk` table cannot be used, you need to go back to the original values in `OriginalFirstThunk` to resolve the imports the old-fashioned way.

Bonus chatter: Binding is of relatively little value nowadays due to address space layout randomization.

[1] And the module timestamp is often not really a timestamp.