LoadLibraryEx(DONT_RESOLVE_DLL_REFERENCES) is fundamentally flawed

devblogs.microsoft.com/oldnewthing/20050214-00

February 14, 2005

Raymond Chen

There is a flag to the `LoadLibraryEx` function called `DONT_RESOLVE_DLL_REFERENCES`. The documentation says,

> If this value is used, and the executable module is a DLL, the system does not call DllMain for process and thread initialization and termination. Also, the system does not load additional executable modules that are referenced by the specified module.
>
> If you are planning only to access data or resources in the DLL, it is better to use LOAD_LIBRARY_AS_DATAFILE.

In my opinion, the above text that "suggests" the `LOAD_LIBRARY_AS_DATAFILE` flag is not strong enough.

`DONT_RESOLVE_DLL_REFERENCES` is a time bomb.

Look carefully at what the flag does and doesn't do. The module is loaded into memory, but its initialization function is not called and no dependent DLLs are loaded. [Typo fixed, 10am.] As a result, you cannot run code from this DLL. (More accurately, if you try, it will crash because the DLL hasn't initialized itself and none of its imports to DLLs have been resolved.) However, unlike the `LOAD_LIBRARY_AS_DATAFILE` flag, the loaded DLL **can** be found by `GetModuleHandle` and can be used by `GetProcAddress`.

Clearly, `GetProcAddress` is a bad idea for something loaded by `DONT_RESOLVE_DLL_REFERENCES`, because as we already noted, you can't run any code from the DLL. What's the point of getting a procedure address from a DLL if you can't call it, after all?

The `GetModuleHandle` part triggers the time bomb.

It is common for somebody to call `GetModuleHandle` to see if a DLL is loaded, and if so, use `GetProcAddress` to get a procedure address and call it. If the DLL had been loaded with `DONT_RESOLVE_DLL_REFERENCES`, both the `GetModuleHandle` will succeed, but the

resulting function will crash when called. The code doing this has no idea that the DLL was loaded with `DONT_RESOLVE_DLL_REFERENCES`; it has no way of protecting itself.

(Note that code that does this is unsafe anyway, because the code that originally loaded the DLL might decide to do a `FreeLibrary` on another thread, causing the code to be ripped out from underneath the first thread. This second problem can be "fixed" by using `GetModuleHandleEx`, which can be instructed to increment the DLL reference count, but that doesn't fix the first problem.)

Even if you used `LoadLibrary` to load the DLL and passed that handle to `GetProcAddress`, you still crash, because the `LoadLibrary` notices that the DLL is already loaded and merely increments the reference count.

```
#include <windows.h>
typedef HINSTANCE (WINAPI *SXA)(HWND, LPCSTR, LPCSTR,
                                LPCSTR, LPCSTR, int);
int __cdecl main(int argc, char* argv[])
{
 if (argc > 1) // set the time bomb
  LoadLibraryEx("shell32.dll", NULL, DONT_RESOLVE_DLL_REFERENCES);
 // victim code runs here
 HINSTANCE h = LoadLibrary("shell32.dll");
 if (h) {
  SXA f = (SXA)GetProcAddress(h, "ShellExecuteA");
  if (f) {
   f(NULL, NULL, "notepad.exe", NULL, NULL, SW_SHOWNORMAL);
  }
  FreeLibrary(h);
 }
}
```

If you run this program with no command line arguments, then everything works just fine: Notepad is launched without incident. However, if you pass a command line argument, this sets the time bomb, and the call to ShellExecuteA crashes in flames because shell32.dll was loaded without having its DLL references resolved.

In other words, `DONT_RESOLVE_DLL_REFERENCES` is fundamentally flawed and should be avoided. It continues to exist solely for backwards compatibility.

Raymond Chen

Follow