The Import Address Table is now write-protected, and what that means for rogue patching

devblogs.microsoft.com/oldnewthing/20221006-07

October 6, 2022

Raymond Chen

For a few years now, the Import Address Table (IAT) has been write-protected.

The import address table is the part of the Windows module (executable or dynamic link library) which records the addresses of functions imported from other DLLs. For example, if your program calls `GetSystemInfo()`, then the executable or DLL will have an entry in its import table that says, "I would like to be able to call the function `GetSystemInfo()` from `kernel32.dll`." When the module is loaded, the system goes and finds that function, obtains its address, and stores it in a table known as the Import Address Table (IAT). When the module needs to call the `GetSystemInfo()` function, it does so by fetching the value from the Import Address Table and calling it.

The Import Address Table is therefore a table filled with function pointers. This makes it an attractive target for attackers looking to achieve remote code injection, since they can overwrite the entry in the Import Address Table (using a write-what-where vulnerability) and redirect a function call to a location of their choosing.

As a defense in depth measure, the Import Address Table is now write-protected. Once the loader has obtained all the function pointers, it write-protects the table to make it harder for an attacker to overwrite it.

This write protection isn't permanent, however. For delay-loaded imports, the Import Address Table holds a pointer to a stub function, so that the first time the module tries to call the imported function, the call is sent to the stub. That stub function looks up the real function and then updates the Import Address Table entry to point to the real function. To perform this update, the delay-load library temporarily makes the Import Address Table read-write, updates the function pointers, and then restores it to read-only status. So there are still small windows of opportunity in which the Import Address Table is unprotected, but the hope is that these windows are quite small and provide enough of an obstacle that attackers won't consider it a fruitful avenue of attack.

This security mitigation was in place in internal builds, and everything looked pretty good. Then the changes rolled out to the Windows Insider Program, and there were many reports of users not being able to sign into their account.

What’s going on?

The users that couldn’t sign in had installed Windows “enhancement tools” that inject themselves into Explorer and rewrite various parts of the operating system in order to implement their various enhancements. One of them accomplished its nefarious task by patching the Import Address Table in order to detour functions that Explorer used, so it could substitute its own alternative. And that enhancement tool assumed that the Import Address Table was permanently read-write.

What was happening is that users would sign in, Explorer would start up, and then the enhancement tool would inject itself into Explorer and try to patch the Import Address Table. This patch would run afoul of the fact that the Import Address Table is now read-only, so they would crash, taking Explorer down with it.

No Explorer means no desktop, no taskbar, no Start menu.

But we didn’t abandon this additional security measure to accommodate these nefarious programs. Users who have such enhancement tools generally understand that the programs are doing sketchy things. When these types of programs are discovered, we put them on the list of apps that are fundamentally incompatible, and what happens next depends on what kind of update introduces the incompatibility.

If the incompatibility is in a major update of Windows, then the program is placed in the compatibility report for that update, and when users attempt to upgrade, they will be warned of the incompatibility. If it’s a free program, then the upgrade installer advises the user that the program will be uninstalled as part of the upgrade. If it’s a paid program, then the upgrade installer will not force-uninstall the program and require users to do it themselves.

User who still miss the features of that enhancement software can go and reinstall it. And then Explorer will crash at logon. But this time, the blame is properly assigned to “the thing you installed most recently”, which is the enhancement software.

Basically, the upgrade process changed the order of operations. What used to be

Working system	
↓	Install Program X
Working system	

↓	Upgrade Windows
Broken system	blame this guy

is now

Working system	
↓	Install Program X
Working system	
↓	Uninstall Program X
Working system	
↓	Upgrade Windows
Working system	
↓	Install Program X
Broken system	blame this guy

On the other hand, things are not so easy if the incompatibility is with an update that comes as part of Patch Tuesday. The patch updater has no opportunity to ask the user questions or gain confirmation. That's by design, because it would be bad to delay the install of a security update. Those things need to get installed as soon as practical. In this case, Windows also has to tweak itself so that the incompatible "system enhancement" program fails in a more benign way. The enhancement program doesn't have to keep working, but it can't crash the system either. This can be done by making some sort of harmless change that nevertheless throws off the enhancement program's patch logic so that it gives up instead of applying a bad patch.

Raymond Chen

Follow