The Old New Thing

Your program loads libraries by their short name and you don't even realize it

5 May 2011 7:00 AM | **20**

In the discussion of the problems that occur if you load the same DLL by both its short and long names, Xepol asserted that any program which loads a DLL by its short name "would have ONLY itself to blame for making stupid, unpredictable, asinine assumptions" and that Windows should "change the loader to NOT load any dll with a short name where the short name and long name do not match."

The problem with this rule, well, okay, one problem is that you're changing the rules after the game has started. Programs have been allowed to load DLLs by their short name in the past; making it no longer possible creates an application compatibility hurdle.

Second, it may not be possible to obtain the long name for a DLL. If the user has access to the DLL but not to one of the directories in the path to the DLL, then the attempt to get its long name will fail. The "directory you can access hidden inside a directory you cannot access" directory structure is commonly used as a less expensive alternative to Access Based Enumeration. Maybe your answer to this is to say that this is not allowed; people who set up their systems this way deserve to lose.

Third, the application often does not control the path being passed to `LoadLibrary`, so it doesn't even know that it's making a stupid, unpredictable, asinine assumption. For example, the program may call `GetModuleFileName` to obtain the directory the application was installed to, then attempt to load satellite DLLs from that same directory. If the current directory was set by passing a short name to `SetCurrentDirectory` (try it: `cd C:\Progra~1`) then the program will unwittingly be making a stupid, unpredictable, asinine decision. (But since the decision is being made consistently, there was no problem up until now.)

When you call `CoCreateInstance`, there is nearly always a `LoadLibrary` happening behind the scenes, because the object you are trying to create is coming from a DLL that somebody else registered. If they registered it with a short name, then any application that wanted to use that object runs afoul of the new rule, even though the application did nothing wrong.

Now, you might argue that this just means that the component provider is the one who made a stupid, unpredicable, asinine assumption. But that may not have been a stupid assumption at the time: 16-bit applications see only short names, so it might have been that that's the only thing they can do. Or the component may have made a conscious decision to register under short names; this was common in the Windows 95 era because file names often passed through components that didn't understand long file names. (Examples: 16-bit applications, backup applications.)

Even if you decide that what was once a reasonable decision is now asinine, you have to get there from here. Do you declare all 16-bit applications asinine (because they can only use those asinine short file name)? Even for the 32-bit components, how do you convince them to switch over? Can you even get them to switch over? Until they do, their component will not be accessible: The shell extension won't be loaded until they fix their registration. A program which anticipated this problem and always loaded DLLs by their short names is now broken. (After all, you had to make an arbitrary decision to use long names exclusively or short names exclusively, and someone may have chosen the other branch of the decision

tree.)

And long name/short name is really just a special case of a single file having multiple names. Other ways of creating multiple names include hard links, symbolic links, and junction points. If a file has two names due to hard links, which one is the "preferred" name and which is the "asinine" name? (And what if the "preferred" name is not available to the user? Suppose you decide that the preferred name is the one that comes first alphabetically. Can I launch a denial-of-service attack by creating a hard link to `C:\Windows\system32\shell32.dll` called `C:\A\A.DLL`? All attempts to link to `C:\Windows\system32\shell32.dll` will fail because it is now the "asinine" name.) Does this rule against loading DLLs by "asinine" names negate part of the functionality of the compatibility paths introduced in Windows Vista (since, under this new rule, you can't use them to load DLLs or execute programs because they entail traversing a symbolic link).

Perhaps these concerns could be addressed in a manner that didn't have all these problems, but it's an awful lot of complexity. In general, simple rules are preferred to complex rules. Especially if the complexity is to address an issue that was not a serious problem to begin with.

Blog - Comment List MSDN TechNet

Comments

John
5 May 2011 7:24 AM
#

> And long name/short name is really just a special case of a single file having multiple names.

Conceptually this is true, but practically there is a difference. Short file names are automatically generated by the operating system and you don't really have any control over them. For operating system components this probably isn't a problem, but for 3rd party software it can be. Foo~1.dll may point to FooA.dll on one machine and FooB.dll on another machine. There is a lot of software out there, and in an 8.3 world there are only so many unique file names.

The MAZZTer
5 May 2011 7:33 AM
#

John: This is why short files names are intended as a compatibility hack for 16-bit Windows apps mainly, and shouldn't be relied on for modern apps (IIRC I had one that assumed C:\Progra~1) was Program Files, except that I had reinstalled Windows and so it was C:\Progra~2). You should always specify the "real" filename when hardcoding (and of course, never hardcode system paths anyway; there are APIs for those for a reason).

Vilx-
5 May 2011 7:34 AM
#

Not being a hardcore Win32 developer, I wonder, what ramifications would it have, if the loader would be modified to translate short filenames to their long counterparts and THEN load the files. Therefore loading the same file by short or long filename would be the same.

Mathieu Garstecki
5 May 2011 8:29 AM
#

@Vilx-: not an expert on that, but I think the loader runs with the rights of the executing user, so the "directory you cannot access" point blocks that approach.

Alex Grigoriev
5 May 2011 8:38 AM
#

Don't you wish x64 should have gotten rid of short names for Win32 subsystem altogether? Like never return them for FindFileFirst, and ignore them for CreateFile.

Alex Grigoriev
5 May 2011 8:40 AM
#

I suggest that in Windows 8 short name creation be disabled by default. Screw those idiots who use them. If there is a big corporation that rins those crappy apps, they can reenable it by policies.

Joshua
5 May 2011 8:40 AM
#

The way to avoid that terrible glitch that lead to this problem is to compare by device & inode numbers. Device number in Windows is in fact a string (X: or \\SERVER\SHARE except for the need to resolve local mappings) while inode number is available via GetFileInformationByHandle.

My compliments to Microsoft for designing 8 byte inode numbers in 1993.

[*Resolving local mappings isn't enough. What if a server has shared DOCUMENTS=C:\Documents and BUDGETS=C:\Documents\Budgets? They are the same device but have different share names. And then there's DFS. -Raymond*]

Clovis
5 May 2011 9:42 AM
#

"And then there's DFS" - I know just what you mean. Their adverts are way more irritating and dispiriting than anything the Windows loader gets up to.

PhilDW
5 May 2011 10:10 AM
#

@Alex, I was thinking the same thing. There's an argument that native 64-bit doesn't need to carry the legacy Win32/16 stuff. Maybe x64 NTFS is only? I think that kind of thing has been done in some cases, but I can't actually think of one right now ;)

Leo Davidson
5 May 2011 3:29 PM
#

You'd be surprised by how much breaks if you turn off short names on your local filesystem.

Short names are still useful when you need an ANSI filename to pass to some dumb component that can't handle Unicode properly. We can wish all we want but the dumb components won't go away and we'll still find times when we need to interface with them.

Some very successful, ubiquitous components/technologies are incredibly poorly written... I'm not sure MS or Windows would look good, even to developers, if it took away the tools that help us work around other people's stupidity.

And most of the time you can forget the short names exist. They don't cause many problems. The only big problem I'm aware of is the change-notification code in Windows randomly sending long or short names, including for deleted files (where if you didn't cache both names of every file you might care about being deleted you have no way to map the name you receive).

Leo Davidson
5 May 2011 3:32 PM
#

Native 64-bit code still needs to pass filenames to 32-bit programs, some of which may require short names.

It would be ridiculous if 64-bit apps had *more* things hidden from, and *fewer* abilities, than their 32-bit counterparts on a 64-bit OS. And the short names would still be there so what would you be solving, exactly? Nothing. You'd just be making life harder for people compiling 64-bit code, and forcing people to proxy out to 32-bit processes more often than they already have to.

Unless you're starting fresh with a whole new OS, I can't see short names going anywhere, sadly.

Joshua
5 May 2011 4:43 PM
#

@Leo: Yeah, I've used that to pass filenames with spaces to a component that didn't like them.

Joshua
5 May 2011 4:44 PM
#

@Leo: Yeah, I've used short names to pass file names with spaces to components that didn't like spaces in filenames.

Mark Sowul
5 May 2011 5:25 PM
#

Short paths can also be used to get around MAX_PATH, so it probably isn't worth getting rid of them.

blogs.msdn.com/.../1275292.aspx

Ivan K
5 May 2011 5:57 PM
#

Nine paragraphs in that blog entry and only ten instances of the word "asinine"?!

C'mon, we can inspire more than that! (It's Friday where I live :)

Neil
6 May 2011 3:21 AM
#

Short names are disabled by default on SBS 2008. (I don't have access to any other comparable versions.) And yes, we have customers who want to be able to launch DOS applications from network paths.

Gabe
6 May 2011 5:47 AM
#

I'm not sure how DFS works, but I would assume that using a combination of 32-bit volume ID and 64-bit file ID would work to detect duplicate DLLs.

640k
6 May 2011 9:13 AM
#

Will this be fixed in time for 128-bit windows or do we have to wait for 256-bit windows?

Joe Dietz
6 May 2011 9:50 AM
#

Short name generation can be disabled in the NTFS file system. Its a good test to run in fact before shipping any application. Install it without short names. If it doesn't work, you are in fact being asinine.

sukru-t
7 May 2011 1:10 AM
#

@Joe Dietz: What about the components yo use? Will you be willing to replace/rewrite them if they depend on sfn functionality?