The path-searching algorithm is not a backtracking algorithm

devblogs.microsoft.com/oldnewthing/20120208-00

February 8, 2012

Raymond Chen

Suppose your PATH environment variable looks like this:

```
C:\dir1;\\server\share;C:\dir2
```

Suppose that you call `LoadLibrary("foo.dll")` intending to load the library at `C:\dir2\foo.dll`. If the network server is down, the `LoadLibrary` call will fail. Why doesn't it just skip the bad directory in the PATH and continue searching?

Suppose the `LoadLibrary` function skipped the bad network directory and kept searching. Suppose that the code which called `LoadLibrary("foo.dll")` was really after the file `\\server\share\foo.dll`. By taking the server down, you have tricked the `LoadLibrary` function into loading `c:\dir2\foo.dll` instead. (And maybe that was your DLL planting attack: If you can convince the system to reject all the versions on the `PATH` by some means, you can then get `LoadLibrary` to look in the current directory, which is where you put your attack version of `foo.dll`.)

This can manifest itself in very strange ways if the two copies of `foo.dll` are not identical, because the program is now running with a version of `foo.dll` it was not designed to use. "My program works okay during the day, but it starts returning bad data when I try to run between midnight and 3am." Reason: The server is taken down for maintenance every night, so the program ends up running with the version in `c:\dir2\foo.dll`, which happens to be an incompatible version of the file.

When the `LoadLibrary` function is unable to contact `\\server\share\foo.dll`, it doesn't know whether it's in the "don't worry, I wasn't expecting the file to be there anyway" case or in the "I was hoping to get that version of the file, don't substitute any bogus ones" case. So it plays it safe and assumes it's in the "don't substitute any bogus ones" and fails the call. The program can then perform whatever recovery it deems appropriate when it cannot load its precious `foo.dll` file.

Now consider the case where there is also a `c:\dir1\foo.dll` file, but it's corrupted. If you do a `LoadLibrary("foo.dll")`, the call will fail with the error `ERROR_BAD_EXE_FORMAT` because it found the `C:\dir1\foo.dll` file, determined that it was corrupted, and gave up. It doesn't continue searching the path for a better version. The path-searching algorithm is not a backtracking algorithm. Once a file is found, the algorithm commits to trying to load that file (a "cut" in logic programming parlance), and if it fails, it doesn't backtrack and return to a previous state to try something else.

Discussion: Why does the `LoadLibrary` search algorithm continue if an invalid directory or drive letter is put on the PATH?

Vaguely related chatter: No backtracking, Part One

Raymond Chen

Follow