The case of the 32-bit program that tries to load a 64-bit DLL

devblogs.microsoft.com/oldnewthing/20170303-00

March 3, 2017

Raymond Chen

One of our escalation engineers wrote up an epic case study of a particular problem that he was able to solve. I'll retell the story here, but redacted, and significantly abbreviated.

A customer reported that when their 32-bit program called `IUpdateSearcher::Begin-Search`, the resulting dialog box was empty. As in the dialog box had no controls on it.

Debugging revealed that the dialog box tried to load the common controls, but instead of loading the 32-bit common controls library, it was trying to load the 64-bit common controls library. Since 32-bit processes cannot load 64-bit DLLs, this attempt fails, and consequently, no dialog controls appear.

The escalation engineer reported that the customer originally had five clients with this problem. Over time, the number of clients with the problem decreased to one. But not because they managed to solve the problem. The number of clients with the problem went down to one because the other four clients canceled their contracts and went with a competitor.

The escalation engineer eventually got access to a system that reproduced the problem (a story within a story that I'm redacting for space), and he found that the behavior on the problem system was very different from those on a clean system. In particular, the test scenario took ten times as long to run on the problem system as it did on a clean system: On the problem system, there was a DLL that was furiously allocating and freeing memory. The customer insisted that the DLL was not the problem, however.

I'm redacting another part of the story where the customer claims to have a virtual machine that reproduces the problem, but turns out that they don't.

Eventually, the customer admits that they just learned that the problem occurs only on systems with that mysterious DLL installed.

Okay, well, it took a long time to get there, but at least we have something that is common to all failures. The mysterious DLL is part of an anti-malware system that bills itself as something like “The most advanced anti-malware platform.” Okay, mister advanced anti-malware, let’s take a look at what you’re up to.

Application compatibility guru extraordinaire Gov Maharaj suggested taking a look at `Wow64DisableWow64FsRedirection` and `Wow64RevertWow64FsRedirection` and to make sure they were matched up. And they were. So much for that theory.

Another week of debugging passes (mixed in with various failed attempts to get a virtual machine that reproduces the problem), and Gov’s suggestion to look at `Wow64DisableWow64FsRedirection` echoes like a flashback scene in a cheesy movie. File system redirection is disabled and re-enabled thousands of times during the running of the scenario. There were many instances of the code in the DLL, and reverse-engineering revealed that they all looked like this:

```
// disable redirection while we do something
void* previousState;
if (Wow64DisableWow64Redirection(&previousState)) {
   ... do something ...
   // All done. Return to the previous state.
   Wow64RevertWow64Redirection(previousState);
}
```

except that there was one case where the code was subtly different:

```
// Code in italics is wrong.
// disable redirection while we do something
void* previousState;
if (Wow64DisableWow64Redirection(&previousState)) {
   ... do something ...
   // All done. Return to the previous state.
   Wow64RevertWow64Redirection(&previousState);
}
```

The anti-malware DLL fell into the trap of the unfortunate choice of data type for the file system redirection cookie. Furthermore, even though the code had a bug, the address of the redirection cookie was lucky enough to look enough like a genuine redirection cookie that it successfully restored the redirection state to what it should have been. But one time out of those thousands, its luck ran out, and it didn’t restore the state properly.

That one call left file system redirection disabled, and thereafter, all the correct code sequences which temporarily disabled redirection would restore the redirection to its previous state, which was now “disabled” rather than the “enabled” it was supposed to be.

To verify that this needle-in-a-haystack situation was the root cause, the escalation engineer patched the DLL to fix the bug, and the test scenario ran to completion successfully.

The customer went back to the vendor of the anti-malware software to get an updated version of the DLL, and that fixed the problem.

Raymond Chen

Follow