Diagnosing a crash in unloaded_something.dll

devblogs.microsoft.com/oldnewthing/20160527-00

May 27, 2016

Raymond Chen

A failure report came in to the shell team because Explorer crashed at shutdown in what the debugger reported as `unloaded_themeui.dll`. Time to dig in.

```
ntdll!RtlpCallVectoredHandlers+0xeb
ntdll!RtlDispatchException+0x81
ntdll!KiUserExceptionDispatch+0x50
<Unloaded_themeui.dll>+0x2bbfbd
0x1fbdebe0
0x1fbdebc0
0x2357ef80

<Unloaded_themeui.dll>+0x2bbfbd:
00007ff8`e384bfbd ??              ???
```

Yup, there's nothing loaded there all right. But let's see what was loaded there before.

```
0:001> lm
...
Unloaded modules:
...
00007ff8`e3590000 00007ff8`e385d000   themeui.dll
00007ff8`e3840000 00007ff8`e385d000   abcdefg.dll
```

So there were two DLLs that used to be loaded at the address that crashed. Which could it be?

```
0:001> !reload /unl themeui.dll
0:001> u 00007ff8`e384bfbd
themeui!ext-ms-win-com-ole32-l1-1-1_NULL_THUNK_DATA_DLA <PERF> (themeui+0x2bffdd):
00007ff8`e384bfbd 40              ???
```

Well, that doesn't look like code. How about abcdefg?

```
0:001> !reload /unl abcdefg.dll
0:001> u 00007ff8`e384bfbd-80
...
abcdefg!AbcdefgImageList::GetClassImageList+0x1f
00007ff8`e384bfb7 ff1593a20000    call    [abcdefg!_imp_SetupDiGetClassImageList]
00007ff8`e384bfbd 85c0            test    eax,eax
```

That looks a lot more promising. What appears to have happened is that `abcdefg.dll` called `SetupDiGetClassImageList`, and while the call was in progress, the DLL got unloaded. When the call to `SetupDiGetClassImageList` finally returned, it returned to an unloaded DLL, which is the source of the crash.

Reconstructing the stack revealed a chain of calls that made sense in the context of `abcdefg.dll`, so this diagnosis is probably correct. (I've anonymized the name of the other DLL to protect the guilty.)

What happened is that during Explorer startup, `abcdefg.dll` registered a wait with the thread pool on an event, and at shutdown it unregisters the wait. But it unregisters with the `UnregisterWait` function. If a callback is running at the time the wait is unregistered, the function returns `ERROR_IO_PENDING`, but nobody checks. The shutdown code proceeds to unload `abcdefg.dll`, and then we are left executing code that was freed.

The code in `abcdefg.dll` needs to handle the case where the callback is still running at the time the wait is unregistered. You can use the `UnregisterWaitEx` function, which lets you pass an event that is set when the callback completes, or pass `INVALID_HANDLE_VALUE` to wait synchronously for the callback to complete before returning.

Raymond Chen

Follow