Debugging walkthrough: Diagnosing an NX exception

devblogs.microsoft.com/oldnewthing/20160211-00

February 11, 2016

Raymond Chen

A colleague of mine asked for help debugging a strange failure. Execution halted because the CPU detected that it was trying to execute data.

```
ABC!__PchSym_ (ABC+0x67be4)
user32!UserCallWinProcCheckWow+0x140
user32!DispatchClientMessage+0xa2
user32!__fnDWORD+0x2d
ntdll!KiUserCallbackDispatcherContinue
user32!ZwUserPeekMessage+0xa
user32!PeekMessageW+0x7f
explorerframe!CExplorerFrame::FrameMessagePump+0x5b
explorerframe!BrowserThreadProc+0x5e
explorerframe!BrowserNewThreadProc+0x3a
explorerframe!CExplorerTask::InternalResumeRT+0x12
explorerframe!CRunnableTask::Run+0xc9
shell32!CShellTaskThread::ThreadProc+0x284
shell32!CShellTaskThread::s_ThreadProc+0x2b
SHCore!_WrapperThreadProc+0x15f
kernel32!BaseThreadInitThunk+0xd
ntdll!RtlUserThreadStart+0x1d

EXCEPTION_RECORD:  (.exr -1)
ExceptionAddress: 00007ffcfd197be4 (ABC+0x67be4)
   ExceptionCode: c0000005 (Access violation)
  ExceptionFlags: 00000000
NumberParameters: 2
   Parameter[0]: 0000000000000008
   Parameter[1]: 00007ffcfd197be4
Attempt to execute non-executable address 00007ffcfd197be4
```

My colleague suspected that a return address got overwritten by some function deeper in the stack, and that caused the instruction pointer to jump to a random module, and the victim module was ABC.

I looked at the crash dump, and came to a different conclusion. The stack is just fine. The problem is that a DLL got unloaded:

```
0:067> lm
...
Unloaded modules:
...
00007ffc`fd140000 00007ffc`fd1ee000   DEF.dll
...
```

After `DEF.dll` got unloaded, `ABC.DLL` got loaded into the same location.

```
0:067> .reload /unl DEF.dll
WARNING: DEF overlaps ABC
```

The problem is that `DEF.dll` unloaded before destroying all its windows. And then its window received a message (in this case, `WM_ACTIVATEAPP`, but you were not expected to know this since it wasn't in the stack trace). The window manager called the window procedure, which now points into the middle of `ABC.dll`. The debugger is correctly reporting that execution halted in the middle of `ABC.dll`.

The next step is to engage the people responsible for `DEF.dll` to figure out why they leaked a window.

Exercise: What command would be useful at this point to help the `DEF.dll` identify the window that they leaked?

Raymond Chen

Follow