Under what conditions can SetFocus crash? Another debugging investigation

devblogs.microsoft.com/oldnewthing/20150529-00

May 29, 2015

Raymond Chen

A customer asked, "Under what conditions can `SetFocus` crash?"

> We have been going through our error reports and are puzzled by this one. The call stack is as follows:
>
> ```
> user32!_except_handler4
> ntdll!ExecuteHandler2@20
> ntdll!ExecuteHandler@20
> ntdll!RtlDispatchException
> ntdll!_KiUserExceptionDispatcher@8
> 0x130862
> user32!UserCallWinProcCheckWow
> user32!__fnDWORD
> ntdll!_KiUserCallbackDispatcher@12
> user32!NtUserSetFocus
> contoso!DismissPopup
> ```
>
> At the point of the crash, the `DismissPopup` function is calling `SetFocus` to restore focus to a window handle that we got from an earlier call to `GetActiveWindow`. Is this safe? We imagine it might crash if the message handler for the window was unloaded from memory without being properly unregistered; are there any other reasons? More to the point, is there any way to avoid the problem (without fixing the root cause of the crash, which we may not be able to do, e.g. if that window was created by third-party code)?
>
> The full dump file can be found on <location>. The password is <xyzzy>.

Indeed, what the customer suspected is what happened, confirmed by the dump file provided.

The code behind the window procedure got unloaded. `UserCallWinProcCheckWow` is trying to call the window procedure, but instead it took an exception. The address doesn't match any loaded or recently-unloaded module probably because it was a dynamically generated thunk, like the ones ATL generates.

There isn't much you can do to defend against this. Even if you manage to detect the problem and avoid calling `SetFocus` in this problematic case, all you're doing is kicking the can further down the road. Your program will crash the next time the window receives a message, which it eventually will. (For example, the next time the user changes a system setting and the `WM_SETTINGCHANGE` message is broadcast to all top-level windows, or the user plugs in an external monitor and the `WM_DISPLAYCHANGE` message is broadcast to all top-level windows.)

Basically, that other component pulled the pin on a grenade and handed it to your thread. That grenade is going to explode sooner or later. The only question is when.

Such is the danger of giving your application an extension model that allows arbitrary third party code to run. The third party code can do good things to make your program more useful, but it can also do bad things to make your program crash.

Raymond Chen

Follow