One way people abused hooks in 16-bit Windows

devblogs.microsoft.com/oldnewthing/20060810-06

August 10, 2006

Raymond Chen

We saw last time how windows hooks were implemented in 16-bit Windows. Even though the `HHOOK` was an opaque data type that should have been treated like a handle, many programs "knew enough to be dangerous" and took advantage of the fact that the `HHOOK` was just a pointer to the previous hook procedure.

The most common way of abusing this knowledge was by unhooking from the windows hook chain the wrong way. Instead of calling the `UnhookWindowsHook` function to unhook a windows hook, they called `SetWindowsHook` again! Specifically, they removed their hook by simply reinstalling the previous hook at the head of the chain:

```
HHOOK g_hhkPrev;
// install the hook
g_hhkPrev = SetWindowsHook(WH_KEYBOARD, MyHookProc);
...
// crazy! uninstall the hook by setting the previous hook "back"
SetWindowsHook(WH_KEYBOARD, g_hhkPrev);
```

This code worked in spite of itself; it's as if two wrongs made a "sort of right". If nobody else messed with the hook chain in between the time the hook was installed and it was subsequently "uninstalled", then reinstalling the hook at the head of the chain did restore the chain variables in the same way they would have been restored if they had uninstalled the hook correctly.

But if somebody else installed their own `WH_KEYBOARD` hook in the meantime, then setting the previous hook "back" would have the effect of not only "uninstalling" the `MyHookProc` but also **all other hooks that were installed in the meantime**. (This is exactly the same problem you have if you aren't careful in how you remove subclassed window procedures.)

I still have no idea why they used this strange technique instead of doing the right thing, which is just swapping out one line of code for another:

```
UnhookWindowsHook(WH_KEYBOARD, MyHookProc);
```

Windows 3.1 introduced the `SetWindowsHookEx` / `CallNextHookEx` model, which doesn't use the external linked list technique but rather manages the hook chain internally. This protected the hook chain from programs that corrupted it by mismanaging the external hook chain, but it meant that when these crazy programs tried to unhook by hooking, they ended up corrupting the **internal** hook chain. Special code had to be written to detect these crazy people and turn their bad call into the correct one so that the hook chain wouldn't get corrupted.

Raymond Chen

Follow