Adventures in application compatibility: The case of the display control panel crash on exit

devblogs.microsoft.com/oldnewthing/20230213-00

February 13, 2023

Raymond Chen

Windows reliability telemetry reported that there were a large number of crashes in the Display control panel. Since these crashes are coming via telemetry and the Windows Error Reporting service, there is no information about what steps are required to reproduce the problem. All we have are crash dumps.

The crash was due to the instruction pointer being in the middle of nowhere. For example, in one dump, it was at address `ffffffff`924bbde0`. Close study shows that this value is suspiciously similar to `00007fff`924bbde0`, which is the address of `ntdll!ButtonWndProc_A`. This tells me that somebody subclassed a button, and then tried to restore the original window procedure, but they messed up and truncated the 64-bit pointer value to a 32-bit signed integer. Bonus insult: Their button is ANSI, not Unicode. It's (checks watch) 2023, get with the program. Not everybody who uses a computer speaks English.

To debug this problem, I had to do some triangulation of the crash dumps to look for a third party component that was common to all (or at least most) of the crashes. Since this was a Display control panel, I focused on the video card information, since video card drivers can provide a custom Display control panel plug-in to show off their driver-specific features.

And I found it.

The custom property sheet that comes with one particular video card has a bug in its `WM_DESTROY` handler: It casts a `WNDPROC` to a 32-bit value, causing the upper 32 bits to be lost.

Here is the reverse-engineered dialog procedure:

```
INT_PTR CALLBACK DialogProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  if (uMsg == WM_INITDIALOG) {
        SetWindowLongPtr(hdlg, GWLP_USERDATA,
                ((PROPSHEETPAGE*)lParam)->lParam);
  }
  MyClass* self = (MyClass*)GetWindowLongPtr(
        hdlg, GWLP_USERDATA);
  return self ? self->RealDialogProc(hdlg, uMsg, wParam, lParam)
              : FALSE;
}
```

And here is the "real" `DLGPROC`:

```
INT_PTR CALLBACK MyClass:RealDialogProc(
    HWND hdlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg)
  {
  ...
  case WM_DESTROY:
    SetWindowLongPtr(GetDlgItem(m_dlg, IDC_SOME_BUTTON),
        GWLP_WNDPROC, (LONG)g_originalWndProc);
   ...
  }
}
```

I suspect this code was originally written as 32-bit code, and the line was

```
    SetWindowLong(GetDlgItem(m_dlg, IDC_SOME_BUTTON),
        GWL_WNDPROC, (LONG)g_originalWndProc);
```

When porting to 64-bit, the `SetWindowLong` becomes `SetWindowLongPtr` to expand the value to 64 bits, and the name of the index changes from `GWL_WNDPROC` to `GWLP_WNDPROC`, with the extra P emphasizing that the value should be passed to `Get` / `SetWindowLongPtr`.

But they forgot to upgrade their cast from `(LONG)` to `(LONG_PTR)`, so they were accidentally truncating their 64-bit value to a sign-extended 32-bit value as part of the restoration.

I went for style points and came up with a one-byte patch to fix the bug.

```
// rbx = (LONG)g_originalWndProc
48631d33540300  movsxd  rbx,dword ptr [contoso+0x39c50]
```

Patch the second byte from `63` to `8b`:

```
// rbx = (LONG_PTR)g_originalWndProc
488b1d33540300  mov     rbx,qword ptr [contoso+0x39c50]
```

It turns out that all the machines that are hitting this bug are running drivers that are over ten years old. The current drivers don't have this bug. In fact, the current drivers don't even have the custom control panel extension! In the time since I originally did this investigation, it appears that people finally got their act together and upgraded their video drivers, because there has been only one recorded occurrence of this crash worldwide in the past 30 days.

Bonus reading: The difference between a junior and senior position at a video card company.