Adventures in application compatibility: The case of the PC-relative indirect jump that reads from nowhere

devblogs.microsoft.com/oldnewthing/20220613-00

June 13, 2022

Raymond Chen

A spike of Explorer crashes occurred with the release of a particular Windows Insider build. The crash looked like this:

```
00007ffb`25cc0000 ff25b29e0600    jmp     qword ptr [00007ffb`25d29eb8]
ds:00007ffb`25d29eb8=????????????????
```

The code did not correspond to any loaded DLL, so this is the result of some sort of code injection. The return address on the stack provided a clue:

```
00007ffb`25cc0000
explorer!TrayUI::RefreshPixieDust+0xab
explorer!TrayUI::WndProc+0x1086
.. window message dispatch functions ..
```

This particular code path runs when you change themes, which isn't something you do frequently, but even in the a Windows Insider population, it happens often enough that the crashes show up as significant in Windows Error Reporting. The `TrayUI::RefreshPixieDust` method crashed at the call to `user32!SetWindowPixueDust`:

```
00000001`4004345c 48ff1555c92700  call    qword ptr [explorer!_imp_SetWindowPixieDust
(00000001`402bfdb8)]
00000001`40043463 0f1f440000      nop     dword ptr [rax+rax]
```

A look at the `SetWindowPixieDust` function shows that it has been detoured!

```
// normal:
00007ffb`25217560 ff25b29e0600    jmp     qword ptr [user32!_imp_NtUserSetWindowPixieDust
(00007ffb`25281418)]

// crash dump:
00007ffb`25217560 48b80000cb25fb7f0000 mov rax, 7ffb25cb0000
00007ffb`2521756a ffe0            jmp     rax
```

At this point, I did some analysis of the crash dumps to see if there was something common to all of the crashes. And I found one: In every crash, a popular third-party "shell enhancement" was running.

So now we know who is doing the detouring. They're probably doing it so that they can "add extra features" to Explorer.

Reminder: Detouring the operating system is not supported. This program is doing unsupported things, but their customers don't know or care.

Let's take a close look at the crashing instruction:

```
00007ffb`25cc0000 ff25b29e0600    jmp     qword ptr [00007ffb`25d29eb8]
ds:00007ffb`25d29eb8=????????????????
```

Notice that the code bytes are identical to the original instruction: `ff 25 b2 9e 06 00` .

The problem is that the instruction they are trying to replicate uses RIP-relative addressing, and they didn't take that into account. The disassembler is doing you a favor and converting the RIP-relative address to an absolute address. A more literal disassembly would be

```
00007ffb`25217560 ff25b29e0600    jmp     qword ptr [rip+69eb2h]
```

With this rewrite, it becomes more obvious that just copying this instruction to another location will alter its behavior, since the relocated instruction will have a different *rip*.

I added to the email thread one of my colleagues who used to work at the company that wrote the software in question, hoping that they might still have contacts at their old company to help get this fixed. My colleague wrote back, "The sad thing here is I wrote the detour code, or at least the 32-bit version of the detour. It looks like they missed a spot when they added 64-bit support."

Anyway, the company issued a fix for their detour code, and the crashes eventually stopped.

Raymond Chen

Follow