Investigating an app compat problem: Part 1: The initial plunge

devblogs.microsoft.com/oldnewthing/20160608-00

June 8, 2016

Raymond Chen

Today we're going to look at an application compatiblity problem. Actually, today, we'll just look at the crash that is the reason why we have an application compatibility problem. We'll then spend the next few days trying to figure out what went wrong. Today's installment will be rather long because I want to go through the entire process of making the initial diagnosis, so that we won't have to spend each day re-establishing context.

I also want to point out that even though this is presented as a straightforward narrative, the actual analysis involved a good number of dead ends and sitting around thinking, "Great, what am I supposed to do now?"

We start with the actual crash.

```
eax=00000001 ebx=31410000 ecx=00000000 edx=40000000 esi=001345a8 edi=1fee0a9c
eip=314259d3 esp=0013443c ebp=0013444c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
contoso!ContosoInitialize+0x4d73:
314259d3 mov     eax,dword ptr [ecx+eax*4]            ds:0023:00000004=????????
```

(Obviously, the symbols are wrong. Which makes sense because we don't have symbols for `contoso.dll`.)

What we have here is an array index operation into an array, but the array is a null pointer. Let's see where that null pointer came from.

```
contoso!ContosoInitialize+0x4d40:
314259a0 push    ebp
314259a1 mov     ebp, esp
314259a3 sub     esp, 10h
314259a6 mov     dword ptr [ebp-10h], ecx
314259a9 mov     eax, dword ptr [ebp+8]
314259ac mov     dword ptr [ebp-8], eax
314259af lea     ecx, [ebp-0Ch]
314259b2 push    ecx
314259b3 lea     edx, [ebp-4]
314259b6 push    edx
314259b7 mov     eax, dword ptr [ebp-8]
314259ba push    eax
314259bb call    contoso!ContosoInitialize+0x4db0 (31425a10)
314259c0 add     esp, 0Ch
314259c3 mov     edx, 1
314259c8 mov     ecx, dword ptr [ebp-0Ch]
314259cb shl     edx, cl
314259cd mov     eax, dword ptr [ebp-4]
314259d0 mov     ecx, dword ptr [ebp-10h]
314259d3 mov     eax, dword ptr [ecx+eax*4]
```

Ah, it came from the `this` pointer. the object itself has an array as its first member, and we're trying to index into it, but the `this` pointer is null, so there is nothing to index into. Where did that null pointer come from? We'll have to go up the stack to see what the caller is passing as `this`.

Here's the stack trace, or at least part of it.

```
ChildEBP RetAddr
0013444c 31426201 contoso!ContosoInitialize+0x4d73
00134478 31427401 contoso!ContosoInitialize+0x55a1
00134484 3142b030 contoso!ContosoInitialize+0x67a1
00134494 3142afdd contoso!ContosoInitialize+0xa3d0
001344c4 314279e0 contoso!ContosoInitialize+0xa37d
001344dc 31427e86 contoso!ContosoInitialize+0x6d80
001344f8 31427c2f contoso!ContosoInitialize+0x7226
00134504 31427bd3 contoso!ContosoInitialize+0x6fcf
00134530 314332b8 contoso!ContosoInitialize+0x6f73
00134554 31432278 contoso!ContosoInitialize+0x12658
00134578 3142dd7a contoso!ContosoInitialize+0x11618
001345a8 3142e326 contoso!ContosoInitialize+0xd11a
00134668 3142e4b5 contoso!ContosoInitialize+0xd6c6
0013468c 3142e5b6 contoso!ContosoInitialize+0xd855
001346b0 31424766 contoso!ContosoInitialize+0xd956
001346bc 31421c0d contoso!ContosoInitialize+0x3b06
001346e0 314c8cba contoso!ContosoInitialize+0xfad
00134790 314b5e08 contoso!ContosoInitialize+0xa805a
001347b0 77c60c6e contoso!ContosoInitialize+0x951a8
001347d0 77c12f17 ntdll!RtlAddAuditAccessAceEx+0x5e
00134820 77c09331 ntdll!RtlActivateActivationContextUnsafeFast+0xd7
001348a0 77c091b4 ntdll!RtlLoadString+0x3c1
001348c4 77c07d68 ntdll!RtlLoadString+0x244
001348e0 77c0bd21 ntdll!LdrGetDllHandleByMapping+0xe8
00134928 77c0b941 ntdll!RtlAcquirePebLock+0x391
00134a74 77c0b615 ntdll!LdrLoadDll+0x3a1
00134af8 751a7f94 ntdll!LdrLoadDll+0x75
00134b3c 751aad06 KERNELBASE!LoadLibraryExW+0x124
00134b64 018aa533 KERNELBASE!LoadLibraryExA+0x26
00134c18 018aa410 setup+0x3a533
```

At this point, you might choose to shudder in horror, because you can see that this DLL is running a ton of code during `DLL_PROCESS_ATTACH`

Once you overcome your initial shock, you can pick off the caller (highlighted in the stack trace as the return address) and take a look at the enclosing function.

```
contoso!ContosoInitialize+0x5570:
314261d0 push    ebp
314261d1 mov     ebp, esp
314261d3 push    0FFFFFFFFh
314261d5 push    offset contoso!ContosoInitialize+0xa5798 (314c63f8)
314261da mov     eax, dword ptr fs:[00000000h]
314261e0 push    eax
314261e1 mov     dword ptr fs:[0], esp
314261e8 sub     esp, 14h
314261eb mov     dword ptr [ebp-20h], ecx // "this"
314261ee mov     eax, dword ptr [ebp+8]
314261f1 push    eax
314261f2 mov     ecx, dword ptr [ebp-20h] // "this"
314261f5 call    contoso!ContosoInitialize+0x5620 (31426280)
314261fa mov     ecx eax
314261fc call    contoso!ContosoInitialize+0x4d40 (314259a0)
```

The null pointer came from the call to mystery member function `31426280`. Let's see how that function calculated its return value. Since the function is long and contains branches, we'll follow the code forward. This is not guaranteed to be accurate, because the contents of memory at the time of the crash may not match the contents of memory at the time the code was executed, but we'll take that chance for now. (Remember, debugging is an exercise in optimism.)

```
contoso!ContosoInitialize+0x5620:
31426280 push    ebp
31426281 mov     ebp, esp
31426283 push    0FFFFFFFFh
31426285 push    offset contoso!ContosoInitialize+0xa57c3 (314c6423)
3142628a mov     eax, dword ptr fs:[00000000h]
31426290 push    eax
31426291 mov     dword ptr fs:[0], esp
31426298 sub     esp, 24h
3142629b mov     dword ptr [ebp-2Ch], ecx // "this"
3142629e mov     eax, dword ptr [ebp-2Ch] // "this"
314262a1 mov     ecx, dword ptr [eax+400h] // read member at offset 0x400
314262a7 cmp     ecx, dword ptr [contoso!ContosoInitialize+0xe5d3c (3150699c)]
314262ad jne     contoso!ContosoInitialize+0x56c0 (31426320)
```

Was this jump taken? Let's find out. First, extract the `this` pointer passed to the mystery member function 31426280, then look at the member at offset `0x400`, then compare it to the value at `3150699c`.

```
0:000> dd 00134478-20 L1
00134458 1fee50f8
0:000> dd 1fee50f8+400 L1
1fee54f8  00000041
0:000> dd 3150699c L1
3150699c  00000000
```

The values do not match, so let's assume the jump was taken.

```
31426320 mov     dword ptr [ebp-10h], 0
31426327 lea     eax, [ebp-10h]             // bonus parameter
3142632a push    eax
3142632b mov     ecx, dword ptr [ebp-2Ch]   // "this"
3142632e mov     edx, dword ptr [ecx+400h]  // the value is 0x41
31426334 push    edx
31426335 call    contoso!ContosoInitialize+0x50b0 (31425d10)
```

Let's follow the money into mystery free function `31425d10`.

```
contoso!ContosoInitialize+0x50b0:
31425d10 push    ebp
31425d11 mov     ebp, esp
31425d13 push    0FFFFFFFFh
31425d15 push    offset contoso!ContosoInitialize+0xa5738 (314c6398)
31425d1a mov     eax, dword ptr fs:[00000000h]
31425d20 push    eax
31425d21 mov     dword ptr fs:[0], esp
31425d28 sub     esp, 10h
31425d2b call    contoso!ContosoInitialize+0x952f3 (314b5f53)
31425d30 push    eax
31425d31 lea     ecx, [ebp-14h]
31425d34 call    contoso!ContosoInitialize+0xfe0 (31421c40)
31425d39 mov     dword ptr [ebp-4], 0
31425d40 mov     eax, dword ptr [ebp+8]
31425d43 push    eax
31425d44 call    dword ptr [contoso!ContosoInitialize+0xa8590 (314c91f0)] //
TlsGetValue -> goes into the variable
31425d4a mov     ecx, dword ptr [ebp+0Ch]
31425d4d mov     dword ptr [ecx], eax
31425d4f mov     edx, dword ptr [ebp+0Ch]
31425d52 cmp     dword ptr [edx], 0
31425d55 jne     contoso!ContosoInitialize+0x5125 (31425d85)
31425d57 call    dword ptr [contoso!ContosoInitialize+0xa840c (314c906c)] //
GetLastError
31425d5d push    eax
31425d5e lea     ecx, [ebp-14h]
31425d61 call    contoso!ContosoInitialize+0x5150 (31425db0)
31425d66 test    eax, eax
31425d68 je      contoso!ContosoInitialize+0x5125 (31425d85)
31425d6a mov     dword ptr [ebp-18h], 0FFFFFFFFh
31425d71 mov     dword ptr [ebp-4], 0FFFFFFFFh
31425d78 lea     ecx, [ebp-14h]
31425d7b call    contoso!ContosoInitialize+0xff0 (31421c50)
31425d80 mov     eax, dword ptr [ebp-18h]
31425d83 jmp     contoso!ContosoInitialize+0x513e (31425d9e)
31425d85 mov     dword ptr [ebp-1Ch], 0
31425d8c mov     dword ptr [ebp-4], 0FFFFFFFFh
31425d93 lea     ecx, [ebp-14h]
31425d96 call    contoso!ContosoInitialize+0xff0 (31421c50)
31425d9b mov     eax, dword ptr [ebp-1Ch]
31425d9e mov     ecx, dword ptr [ebp-0Ch]
31425da1 mov     dword ptr fs:[0], ecx
31425da8 mov     esp, ebp
31425daa pop     ebp
31425dab ret
```

There are two exit paths for this function; one returns `-1` and the other returns `0`. Let's see if we can figure out which path got executed. The execution is straight-line until we reach this decision:

```
31425d39 mov     dword ptr [ebp-4], 0
31425d40 mov     eax, dword ptr [ebp+8]     // Parameter 0 = 0x00000041
31425d43 push    eax
31425d44 call    dword ptr [contoso!ContosoInitialize+0xa8590 (314c91f0)] //
TlsGetValue
31425d4a mov     ecx, dword ptr [ebp+0Ch]   // Parameter 1 (the bonus parameter)
31425d4d mov     dword ptr [ecx], eax       // Save the TLS value into parameter 1
31425d4f mov     edx, dword ptr [ebp+0Ch]   // Parameter 1
31425d52 cmp     dword ptr [edx], 0         // Was the TLS value zero?
31425d55 jne     contoso!ContosoInitialize+0x5125 (31425d85)
```

I was able to figure out that the called function was `TlsGetValue` by looking at the value in the dump file.

```
0:000> ln poi 314c91f0
kernel32!TlsGetValue:
7603a1f0 mov     edi,edi
```

The branch is taken if the value in the TLS slot is nonzero. So is it? Let's manually re-execute the call to `TlsGetValue`.

```
KERNELBASE!TlsGetValue:
751b6ed0 mov     edi, edi
751b6ed2 push    ebp
751b6ed3 mov     ebp, esp
751b6ed5 mov     ecx, dword ptr fs:[18h] // get the TEB
751b6edc mov     eax, dword ptr [ebp+8] // tlsIndex (0x00000041)
751b6edf mov     dword ptr [ecx+34h], 0 // SetLastError(NOERROR);
751b6ee6 cmp     eax, 40h
751b6ee9 jae     KERNELBASE!TlsGetValue+0x26 (751b6ef6) // Jump taken
...
751b6ef6 cmp     eax,440h
751b6efb jae     KERNELBASE!TlsGetValue+0x42 (751b6f12) // Jump not taken
751b6efd mov     ecx,dword ptr [ecx+0F94h]
751b6f03 test    ecx,ecx
751b6f05 je      KERNELBASE!TlsGetValue+0x4c (751b6f1c)

0:000> dd @$teb+f94 L1
7ffdff94  05f5bf30
```

The value is nonzero, so the jump is not taken.

```
751b6f07 mov     eax,dword ptr [ecx+eax*4-100h]
751b6f0e pop     ebp
751b6f0f ret     4
0:000> dd 05f5bf30+41*4-100 L1
05f5bf34  00000000
```

The TLS value is zero.

Okay, let's go back to the function that called `TlsGetValue` and see what it does when the TLS value is zero.

```
31425d52 cmp     dword ptr [edx], 0         // Was the TLS value zero?
31425d55 jne     contoso!ContosoInitialize+0x5125 (31425d85) // Jump not taken
31425d57 call    dword ptr [contoso!ContosoInitialize+0xa840c (314c906c)] //
GetLastError
31425d5d push    eax                        // NOERROR
31425d5e lea     ecx,[ebp-14h]
31425d61 call    contoso!ContosoInitialize+0x5150 (31425db0)
31425d66 test    eax,eax
31425d68 je      contoso!ContosoInitialize+0x5125 (31425d85)
```

We don't pay attention to `GetLastError` because it doesn't affect flow control. The thing that does affect flow control is function `31425db0`, so let's see what it does.

```
contoso!ContosoInitialize+0x5150:
31425db0 push    ebp
31425db1 mov     ebp,esp
31425db3 push    ecx
31425db4 mov     dword ptr [ebp-4],ecx
31425db7 mov     eax,dword ptr [ebp-4]
31425dba mov     ecx,dword ptr [ebp+8]
31425dbd mov     dword ptr [eax+4],ecx
31425dc0 mov     eax, dword ptr [ebp+8]
31425dc3 mov     esp, ebp
31425dc5 pop     ebp
31425dc6 ret     4
```

The function returns its input parameter. (It does other stuff, but we aren't interested in that part yet.) Therefore, since we passed `NOERROR` as the error code, it also returns `NOERROR`, which is zero. We also saw that the TLS value zero is stored into the bonus parameter, which was `[ebp-10h]`. Armed with this information, we can continue our analysis back to the caller:

```
31425d66 test    eax,eax
31425d68 je      contoso!ContosoInitialize+0x5125 (31425d85) // jump taken
```

And we saw from the analysis earlier that once you hit the code path at `31425d85`, the function will return zero.

Now we can unwind back to the caller and see what happens when function `31425d10` returns zero.

```
31426335 call    contoso!ContosoInitialize+0x50b0 (31425d10) // Returns zero
3142633a add     esp, 8
3142633d cmp     eax, 0FFFFFFFFh
31426340 jne     contoso!ContosoInitialize+0x56e9 (31426349) // jump not taken
31426342 xor     eax, eax                            // return zero
31426344 jmp     contoso!ContosoInitialize+0x5783 (314263e3)

314263e3 mov     ecx, dword ptr [ebp-0Ch]
314263e6 mov     dword ptr fs:[0], ecx
314263ed mov     esp, ebp
314263ef pop     ebp
314263f0 ret
```

Okay, so now we know where the null pointer came from: The null pointer was stored in the TLS slot, and this function returns whatever is in the TLS slot.

That is a superficial analysis of why the program crashed. It doesn't really tell us what happened in the operating system that induced the crash; we'll have to dig in deeper.

The story continues next time.

(function () { var arrowAngle = Math.PI / 12; var arrowLength = 10; function vec2(x, y) { this.x = x; this.y = y; } vec2.prototype.add = function (v) { return new vec2(this.x + v.x, this.y + v.y); }; vec2.prototype.sub = function (v) { return new vec2(this.x – v.x, this.y – v.y); } vec2.prototype.magnitude = function () { return Math.sqrt(this.x * this.x + this.y * this.y); }; vec2.prototype.scale = function (scale) { return new vec2(this.x * scale, this.y * scale); }; vec2.prototype.rotate = function (theta) { var sth = Math.sin(theta); var cth = Math.cos(theta); return new vec2(this.x * cth – this.y * sth, this.x * sth + this.y * cth); }; function directionVector(v0, v1, l) { var d = v1.sub(v0); return d.scale((l || 1) / (d.magnitude() || 1)); } function drawArrowHead(ctx, v0, v1) { var u = directionVector(v1, v0, arrowLength); ctx.beginPath(); ctx.moveTo(v1.x, v1.y); var v = v1.add(u.rotate(arrowAngle)); ctx.lineTo(v.x, v.y); v = v1.add(u.rotate(-arrowAngle)); ctx.lineTo(v.x, v.y); ctx.lineTo(ctx, v1.x, v1.y); ctx.fill(); } function getAttachment(ctx, element, attach) { var v = new vec2(element.offsetLeft – ctx.canvas.offsetLeft, element.offsetTop – ctx.canvas.offsetTop); var mag = parseInt(attach) * 10; if (isNaN(mag)) mag = 10; var dv = new vec2(-mag, -mag); if (attach.indexOf(“S”) >= 0) { v.y += element.offsetHeight; dv.y = mag; } else if (attach.indexOf(“N”) = 0) { v.x += element.offsetWidth; dv.x = mag; } else if (attach.indexOf(“W”) < 0) { v.x += element.offsetWidth / 2; dv.x = 0; } return [v, v.add(dv)]; } function onresize() { var thisPost = document.getElementById("appCompatPart1"); var canvas = thisPost.getElementsByTagName("canvas")[0]; if (!canvas.getContext) return; canvas.width = thisPost.offsetWidth; canvas.height = thisPost.offsetHeight; var ctx = canvas.getContext("2d"); ctx.fillStyle = "#404040"; Array.prototype.forEach.call(thisPost.querySelectorAll("[data-to]"), function (source) { var ids = source.getAttribute("data-to"); ids.split(" ").forEach(function (id) { var opt = id.split(":"); // target:connector:attachStart:attachEnd var target =

```
thisPost.querySelector("#" + opt[0]); var vStart = getAttachment(ctx, source, opt[2]); var vEnd = getAttachment(ctx, target, opt[3]); var v0 = vStart[0]; var v1 = vStart[1]; var v2 = vEnd[1]; var v3 = vEnd[0]; ctx.beginPath(); ctx.moveTo(v0.x, v0.y); switch (opt[1]) { case "L": ctx.lineTo(v3.x, v3.y); v2 = v0; break; case "C": ctx.bezierCurveTo(v1.x, v1.y, v2.x, v2.y, v3.x, v3.y); break; } ctx.stroke(); drawArrowHead(ctx, v2, v3); }); }); } if (window.addEventListener) { window.addEventListener("resize", onresize); onresize(); } })();
```

Raymond Chen

Follow