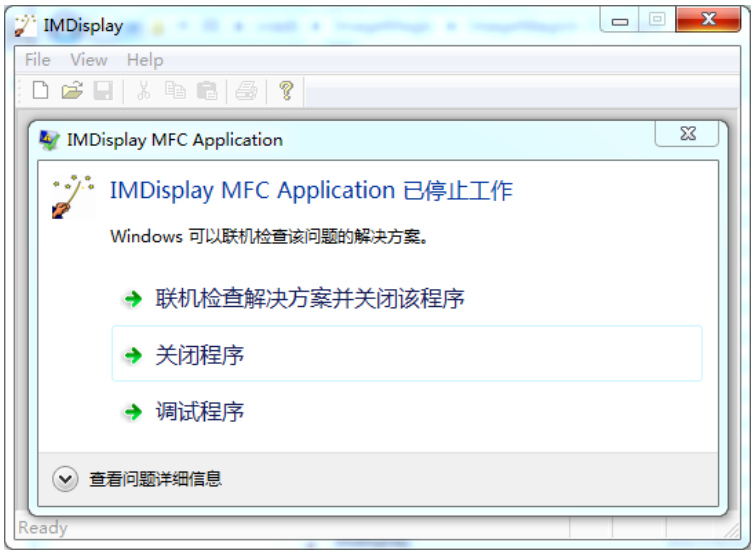
##Overview

For the windows binary release ImageMagick-7.0.5-6, this is a divide-by-zero bug in CORE_RL_openjpeg_!opj_write_tile+0x29fb when opening a crafted file with IMDisplay MFC Application.



##Crash Info

```
(1984.bf4): Integer divide-by-zero - code c0000094 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** WARNING: Unable to verify checksum for D:\IIE\crash\ImageMagic\ImageMagick-7.0.5-Q16\CORE_RL_openjpeg_.dll
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for D:\IIE\crash\ImageMagic\ImageMagick
CORE_RL_openjpeg_!opj_write_tile+0x29fb:
000007fe`dd33b61b f7fb          idiv    eax,ebx
```

```
0:000> r rbx
rbx=0000000000000000
```

##Crash Stack (main part)

```
0:000> kp
Child-SP          RetAddr           Call Site
00000000`002c6aa0 000007fe`dd340051 CORE_RL_openjpeg_!opj_write_tile+0x29fb
00000000`002c6b00 000007fe`dd342360 CORE_RL_openjpeg_!opj_write_tile+0x7431
00000000`002c6bb0 000007fe`dd32a818 CORE_RL_openjpeg_!opj_write_tile+0x9740
00000000`002c6c10 000007fe`dd32aa7e CORE_RL_openjpeg_!opj_image_tile_create+0x56f8
00000000`002c6c70 000007fe`dd32b837 CORE_RL_openjpeg_!opj_image_tile_create+0x595e
00000000`002c6d20 000007fe`dd32a379 CORE_RL_openjpeg_!opj_image_tile_create+0x6717
00000000`002c6d60 000007fe`dd3618b2 CORE_RL_openjpeg_!opj_image_tile_create+0x5259
00000000`002c6d90 000007fe`ddf7bb54 IM_MOD_RL_jp2_+0x18b2
00000000`002c8ed0 000007fe`e0fefd4f CORE_RL_MagickCore_!ReadImage+0x2b4
00000000`002ce020 00000001`3f8d5c8b CORE_RL_Magick___!Magick::Image::read+0x4f
00000000`002ce050 00000001`3f8d609c IMDisplay+0x5c8b
00000000`002ce110 000007fe`dd748cf5 IMDisplay+0x609c
```

##Analysis

In the crash function: opj_write_tile+0x2710

```
118         if ( v18 < v16 )
119         {
120           v19 = *(_DWORD *)v17;                 // v19=2
121           v20 = *(_DWORD *)(v1 + 216);
122           v21 = *(_DWORD *)(v1 + 220);
123           v22 = *(_DWORD *)(v1 + 236);
124           v23 = v16 - v18 - 1;
125           v24 = *(_DWORD *)v17;
126           v25 = *(_QWORD *)(v17 + 16) + 16i64 * v18;
127           v26 = *(_DWORD *)(v17 + 4);
128           v27 = v24 << v23;                                  v19 << v30 = 2 << 31 = 0
129           v37 = v27;                                         => divide by zero
130           v41 = v26 << v23;
131           v28 = (v27 + v20 - 1) / v27;
132           v29 = ((v26 << v23) + v21 - 1) / (v26 << v23);
133           v39 = (v27 + *(_DWORD *)(v1 + 224) - 1) / v27;
134           v40 = ((v26 << v23) + *(_DWORD *)(v1 + 228) - 1) / (v26 << v23);
135           v30 = *(_DWORD *)v25 + v23;           // v30=0x1f=31
136           v38 = *(_DWORD *)v25;
137           v31 = *(_DWORD *)(v25 + 4) + v23;
138           v42 = *(_DWORD *)(v25 + 4);
139           if ( !(v22 % (v26 << (*(_BYTE *)(v25 + 4) + v23))) || v22 == v21 && (v29 << v23) % (1 << v31) )
140           {
141             v32 = *(_DWORD *)(v1 + 232);
142             if ( !(*(_DWORD *)(v1 + 232) % (v19 << v30)) || v32 == v20 && (v28 << v23) % (1 << v30) )
```

```
loc_7FEDD33B60B:
mov     r11d, [r8+0E8h]
mov     ecx, r14d
shl     ebx, cl
mov     eax, r11d
cdq
idiv    ebx
test    edx, edx
jz      short loc_7FEDD33B646
```

If the second operand of *shl* operation is big enough, it is easy to make the result become zero. Then the result is used as a divisor to get the mod value, where the crash happens!

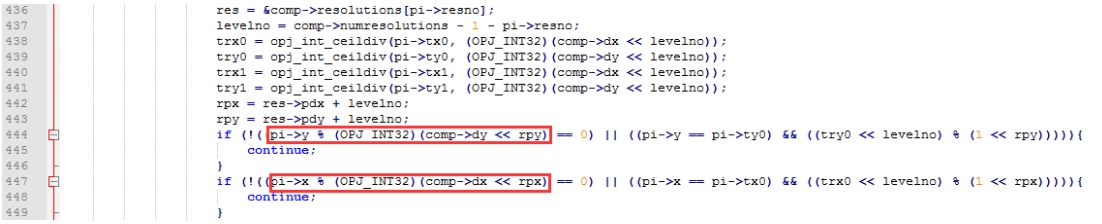
The calling stack is quite large, so I do not go to analysis them one by one. Briefly, the crash happens when ImageMagick reads and parses the image, so the key value leading to crash should be related to the input file.

##Suggested Patch

To give possible patch, I try to locate the crash in source file.
Source_code\openjpeg\src\lib\openjp2\pi.c:

```
1867 OPJ_BOOL opj_pi_next(opj_pi_iterator_t * pi) {
1868     switch (pi->poc.prg) {
1869         case OPJ_LRCP:
1870             return opj_pi_next_lrcp(pi);
1871         case OPJ_RLCP:
1872             return opj_pi_next_rlcp(pi);
1873         case OPJ_RPCL:
1874             return opj_pi_next_rpcl(pi);
1875         case OPJ_PCRL:
1876             return opj_pi_next_pcrl(pi);
1877         case OPJ_CPRL:
1878             return opj_pi_next_cprl(pi);
1879         case OPJ_PROG_UNKNOWN:
1880             return OPJ_FALSE;
1881     }
1882
1883     return OPJ_FALSE;
1884 }
1885
```

The last three functions have the similar structure. I will show one of them: opj_pi_next_pcrl()



```
436                 res = &comp->resolutions[pi->resno];
437                 levelno = comp->numresolutions - 1 - pi->resno;
438                 trx0 = opj_int_ceildiv(pi->tx0, (OPJ_INT32)(comp->dx << levelno));
439                 try0 = opj_int_ceildiv(pi->ty0, (OPJ_INT32)(comp->dy << levelno));
440                 trx1 = opj_int_ceildiv(pi->tx1, (OPJ_INT32)(comp->dx << levelno));
441                 try1 = opj_int_ceildiv(pi->ty1, (OPJ_INT32)(comp->dy << levelno));
442                 rpx = res->pdx + levelno;
443                 rpy = res->pdy + levelno;
444                 if (!((pi->y % (OPJ_INT32)(comp->dy << rpy)) == 0) || ((pi->y == pi->ty0) && ((try0 << levelno) % (1 << rpy))))){
445                     continue;
446                 }
447                 if (!((pi->x % (OPJ_INT32)(comp->dx << rpx)) == 0) || ((pi->x == pi->tx0) && ((trx0 << levelno) % (1 << rpx))))){
448                     continue;
449                 }
```

So, before calculating the mod value, we should first check whether the divisor equals to zero. If so, print error info and finish the program.

##Author

Name: Jiaqi Peng of VARAS@IIE
Organization: IIE(http://iie.ac.cn)