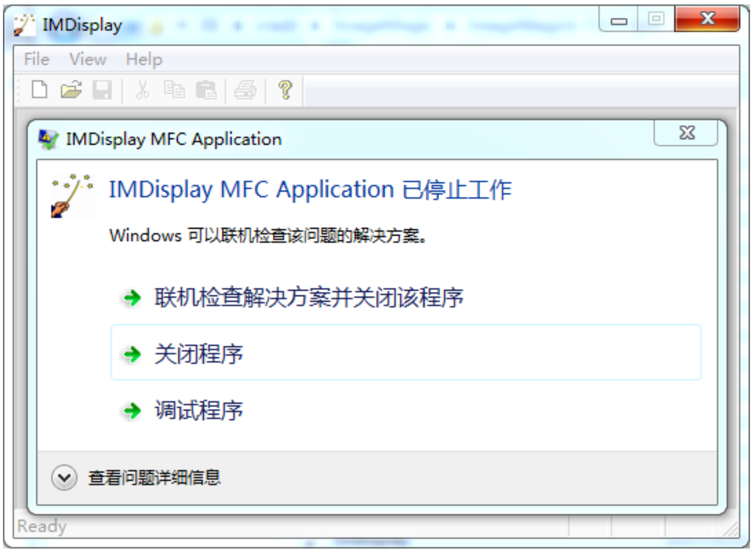
##Overview

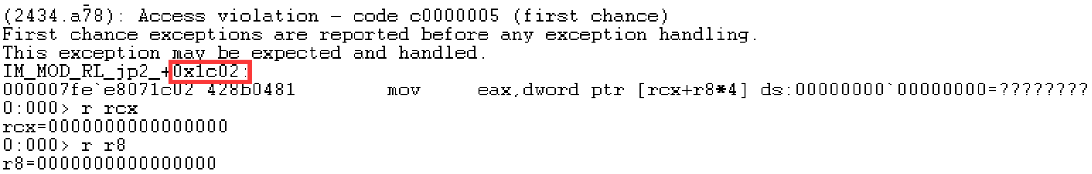
For the windows binary release ImageMagick-7.0.5-6, this is a null-pointer deference bug in IM_MOD_RL_jp2_+0x1c02 when opening a crafted file with IMDisplay MFC Application.



IMDisplay MFC Application 已停止工作

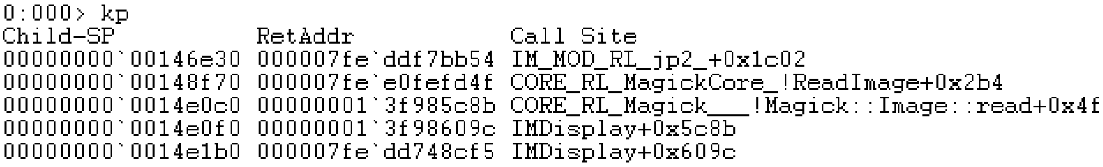
Windows 可以联机检查该问题的解决方案。

##Crash Info



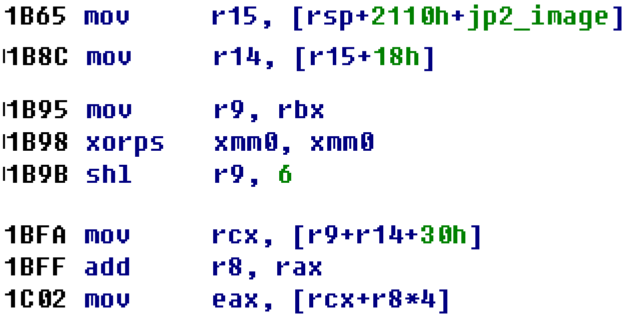
```
(2434.a78): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
IM_MOD_RL_jp2_+0x1c02:
000007fe`e8071c02 428b0481        mov     eax,dword ptr [rcx+r8*4] ds:00000000`00000000=????????
0:000> r rcx
rcx=0000000000000000
0:000> r r8
r8=0000000000000000
```

##Crash Stack (main part)



```
0:000> kp
Child-SP          RetAddr           Call Site
00000000`00146e30 000007fe`ddf7bb54 IM_MOD_RL_jp2_+0x1c02
00000000`00148f70 000007fe`e0fefd4f CORE_RL_MagickCore_!ReadImage+0x2b4
00000000`0014e0c0 00000001`3f985c8b CORE_RL_Magick___!Magick::Image::read+0x4f
00000000`0014e0f0 00000001`3f98609c IMDisplay+0x5c8b
00000000`0014e1b0 000007fe`dd748cf5 IMDisplay+0x609c
```

##Analysis



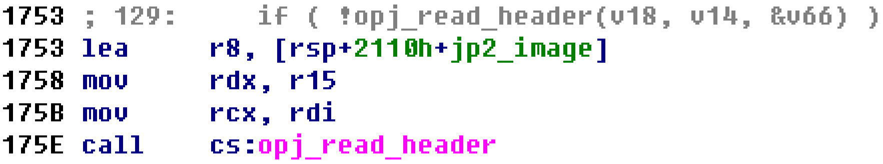
```
1B65 mov     r15, [rsp+2110h+jp2_image]
1B8C mov     r14, [r15+18h]

1B95 mov     r9, rbx
1B98 xorps   xmm0, xmm0
1B9B shl     r9, 6

1BFA mov     rcx, [r9+r14+30h]
1BFF add     r8, rax
1C02 mov     eax, [rcx+r8*4]
```

Crash happened in loc:1C02, where both rcx and r8 = 0. Due to rcx is the base address and r8 is the offset, we infer that the root cause is rcx=0.

rcx = *(r9+r14+0x30) r9 = rbx<<6 = 0<<6 = 0 r14 = *(r15+0x18)

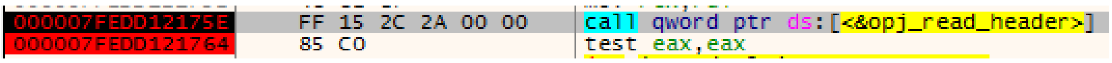
⇨ **rcx = *(*(r15+0x18) + 0x30)**

r15 is some value in stack, so we analysis the function from top to check when the stack value is set. **opj_read_header()** is the first such function and argument r8 is the address of r15.



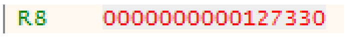
```
1753 ; 129:     if ( !opj_read_header(v18, v14, &v66) )
1753 lea      r8, [rsp+2110h+jp2_image]
1758 mov      rdx, r15
175B mov      rcx, rdi
175E call     cs:opj_read_header
```

Then we debug to see when and how r15 is changed.

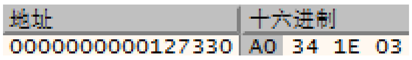


```
000007FEDD12175E   FF 15 2C 2A 00 00   call qword ptr ds:[<&opj_read_header>]
000007FEDD121764   85 C0               test eax,eax
```

Before calling, r8=0x127330, and the value stored in r8 is 0x31E34A0.

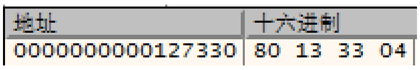


```
R8    0000000000127330
```

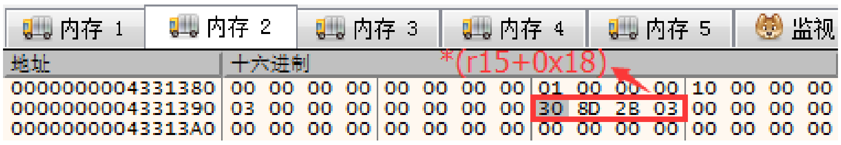


```
地址              十六进制
0000000000127330 A0 34 1E 03
```

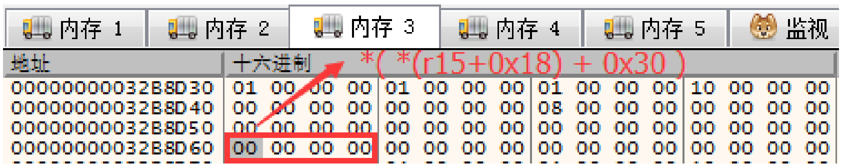
After calling, the value in r8 becomes 0x4331380 => r15=0x4331380



```
地址              十六进制
0000000000127330 80 13 33 04
```



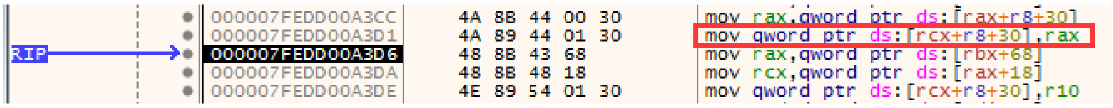
```
地址              十六进制
0000000004331380 00 00 00 00 00 00 00 00 01 00 00 00 10 00 00 00
0000000004331390 03 00 00 00 00 00 00 00 30 8D 2B 03 00 00 00 00
00000000043313A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



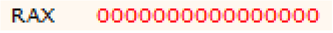
```
地址              十六进制
00000000032B8D30 01 00 00 00 01 00 00 00 01 00 00 00 10 00 00 00
00000000032B8D40 00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00
00000000032B8D50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000032B8D60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

From here, we can know after calling opj_read_header(), rcx=*(*(r15+0x18)+0x30)=0.

Next, we put a hardware write breakpoint in 0x32B8D60 to check whether value in the address will be changed later. Then, we hit the breakpoint here:



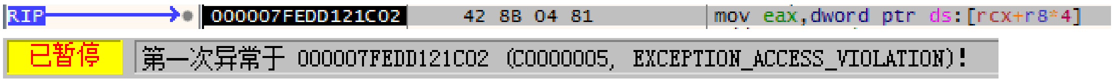
```
      000007FEDD00A3CC   4A 8B 44 00 30   mov rax,qword ptr ds:[rax+r8+30]
      000007FEDD00A3D1   4A 89 44 01 30   mov qword ptr ds:[rcx+r8+30],rax
RIP → 000007FEDD00A3D6   48 8B 43 68      mov rax,qword ptr ds:[rbx+68]
      000007FEDD00A3DA   48 8B 48 18      mov rcx,qword ptr ds:[rax+18]
      000007FEDD00A3DE   4E 89 54 01 30   mov qword ptr ds:[rcx+r8+30],r10
```



```
RAX    0000000000000000
```

But, the value written into 0x32B8D60 is still zero.

With continue execution, we then just come to the exception point!



```
RIP → 000007FEDD121C02   42 8B 04 81   mov eax,dword ptr ds:[rcx+r8*4]
已暂停  第一次异常于 000007FEDD121C02 (C0000005, EXCEPTION_ACCESS_VIOLATION)!
```

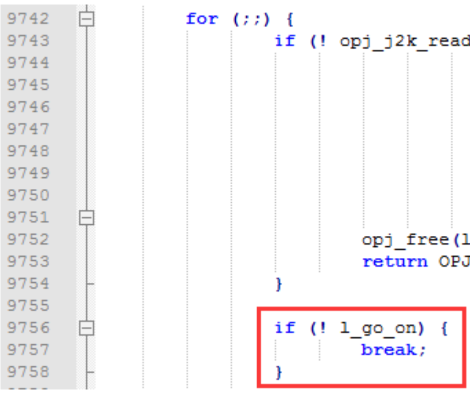
From the analysis above, there are two chances to write data to the base address rcx:

1. opj_read_header(): initialize rcx to zero
2. change rcx to some value, which happens when calling opj_decode() in the crash function

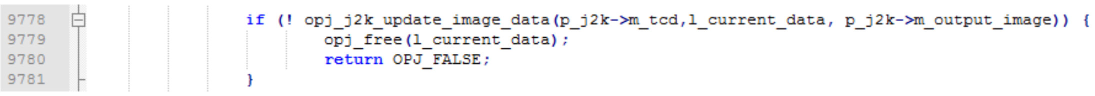
For the second function opj_decode(), we get the calling sequence:

opj_decode()@openjpeg/src/lib/openjp2/openjpeg.c ->
opj_j2k_decode() @openjpeg/src/lib/openjp2/j2k.c ->
opj_j2k_exec() ->
opj_j2k_decode_tiles() ->
opj_j2k_update_image_data() which will write data to the target address

However, in opj_j2k_decode_tiles():



```
        for (;;) {
            if (! opj_j2k_read_tile_header( p_j2k,
                                            &l_current_tile_no,
                                            &l_data_size,
                                            &l_tile_x0, &l_tile_y0,
                                            &l_tile_x1, &l_tile_y1,
                                            &l_nb_comps,
                                            &l_go_on,
                                            p_stream,
                                            p_manager)) {
                    opj_free(l_current_data);
                    return OPJ_FALSE;
            }

            if (! l_go_on) {
                    break;
            }
```



```
                if (! opj_j2k_update_image_data(p_j2k->m_tcd,l_current_data, p_j2k->m_output_image)) {
                        opj_free(l_current_data);
                        return OPJ_FALSE;
                }
```

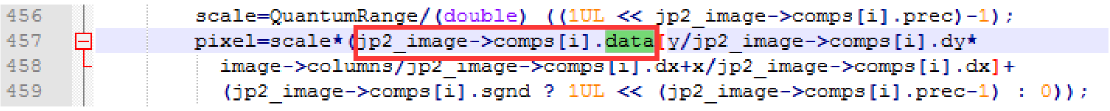
The loop will terminate because l_go_on=False and function opj_j2k_update_image_data() is never called, so the target address is always be zero.

What’s more, when the loop terminates, the function returns OPJ_TRUE, which will give no false information back to its caller function. When back to crash function, the return status of opj_decode() will be TRUE and then we will come to the deference of the target address, which causes crash.

##Suggested Patch

To give possible patch, I try to locate the crash in source file.

Crash function is ReadJP2Image() @ Source_code\ImageMagick\coders\jp2.c and crash happens in line 457 when access jp2_image->comps[0].data[]:



```
456          scale=QuantumRange/(double) ((1UL << jp2_image->comps[i].prec)-1);
457          pixel=scale*(jp2_image->comps[i].data[y/jp2_image->comps[i].dy*
458            image->columns/jp2_image->comps[i].dx+x/jp2_image->comps[i].dx]+
459            (jp2_image->comps[i].sgnd ? 1UL << (jp2_image->comps[i].prec-1) : 0));
```

Here, the base address: jp2_image->comps[0].data = 0

So, the patch can be: **before line 457 add a check: whether jp2_image->comps[i].data!=0**.

Or, in the function opj_j2k_decode_tiles(), when the loop terminates without calling opj_j2k_update_image_data() even once, we should give some error information that can be transferred to upper functions, then some error handlers can be taken before line 457.

##Author

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