Analysis of Use-After-Free Vulnerability (CVE-2016-4119) in Adobe Acrobat and Reader

(/2016/06/06/analysis-of-use-after-free-vulnerability-cve-2016-4119-in-adobe-acrobat-and-reader)

by **Kai Lu and Kushal Arvind Shah** | Jun 06, 2016 | Filed in: Security Research (/category/security-research)

Summary

Recently, Adobe patched some security vulnerabilities (https://helpx.adobe.com/security/products/acrobat/apsb16-14.html) in Adobe Acrobat and Reader. One of them is a use-after-free vulnerability (CVE-2016-4119) discovered by Fortinet's FortiGuard Labs. In this blog, we want to share our analysis of this vulnerability.

Proof of Concept

This vulnerability can be reproduced by opening the PDF file "PoC_decrypt.pdf" with Adobe Reader DC. When opened, AcroRd32.exe crashes, and the crash information shows the following:

```
(28d8.110): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=26ace5e0 ebx=07887004 ecx=25b71fd8 edx=00000001 esi=1cc81d14 edi=26bcd81c
eip=6021d2e1 esp=26ace5cc ebp=26ace5ec iopl=0 nv up ei pl nz na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00010206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661:
6021d2e1 8b01 mov eax,dword ptr [ecx] ds:002b:25b71fd8=????????

1:027> kb
ChildEBP RetAddr Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
26ace5ec 6021d2a3 1ca54428 26ace624 6056bd31 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661
26ace5f8 6056bd31 00000001 e3163325 26bcd81c AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24623
26ace624 6056bf9f 1ca544e8 26ace63c 601253d0 AcroRd32_60000000!AX_PDXlateToHostEx+0x1f4efa
26ace630 601253d0 00000001 26ace668 0799e13b AcroRd32_60000000!AX_PDXlateToHostEx+0x1f5168
26ace63c 0799e13b 1ca54428 e317e5fe 26bcd81c AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x7be7f
26ace668 0799e0ff 0788484c 1ca544e8 26bcd81c AGM!AGMInitialize+0x658df
26ace694 07885410 00000001 26ace70c 07948e77 AGM!AGMInitialize+0x658a3
26ace6a0 07948e77 26bcd81c 0cff3ba8 0797701a BIB!BIBInitialize4+0x3cd2
26ace70c 07885410 00000001 26acef84 07948e77 AGM!AGMInitialize+0x1061b
26ace718 07948e77 26bcdc18 00000004 07f697f2 BIB!BIBInitialize4+0x3cd2
26acefd8 0798d491 26acf07c 26acf05c 00000009 AGM!AGMInitialize+0x1061b
26acf024 07a378c0 1ca62834 00000000 26acf05c AGM!AGMInitialize+0x54c35
26acf060 07968be6 26acf1e8 26acf1b0 00000000 AGM!AGMGetVersion+0x77cbc
26acf0c4 07882479 0789a000 26acf5a8 26acf5c8 AGM!AGMInitialize+0x3038a
26acf0dc 07886bf0 e317f2d0 0789a000 0cff38b4 BIB!BIBInitialize4+0xd3b
26acf654 07f699bf 07f69a50 26acf86c 26acfe4c BIB!BIBGetGetProcAddress+0x1163
26acf76c 07c6c87b 00000000 0798ab66 07988f41 MSVCR120!CallCatchBlock+0x140 [f:\dd\vctools\crt\crtw32\eh\frame.cpp @ 1455]
26acf8e0 07961ba4 00000000 16020760 07961b79 AGM!AGMGetVersion+0x2acc77
26acf8ec 07961b79 00000000 0795ea3b 26acf944 AGM!AGMInitialize+0x29348
26acf8f4 0795ea3b 26acf944 e317fa92 00000000 AGM!AGMInitialize+0x2931d
00000000 00000000 00000000 00000000 00000000 AGM!AGMInitialize+0x261df

1:027> !heap -p -a ecx
    address 25b71fd8 found in
     _DPH_HEAP_ROOT @ 2e21000
     in free-ed allocation ( DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
     259c17b8:         25b71000         2000
     112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
     7df4251c ntdll!RtlDebugFreeHeap+0x0000002f
     7defb2a2 ntdll!RtlpFreeHeap+0x0000005d
     7dea2ce5 ntdll!RtlFreeHeap+0x00000142
     7dd714bd kernel32!HeapFree+0x00000014
     07f5ecfa MSVCR120!free+0x0000001a
     601fe7e8 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00005b68
     6056e29b AcroRd32_60000000!AX_PDXlateToHostEx+0x001f7464
     601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
     0798d5ff AGM!AGMInitialize+0x00054da3

1:027> u
    AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661:
    6021d2e1 8b01          mov         eax,dword ptr [ecx]
    6021d2e3 52          push         edx
    6021d2e4 ff10          call         dword ptr [eax]
    6021d2e6 83a6c800000000     and     dword ptr [esi+0C8h],0
    6021d2ed 8b8618010000     mov     eax,dword ptr [esi+118h]
    6021d2f3 85c0          test         eax,eax
    6021d2f5 742a          je         AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x246a1 (6021)


    6021d2f7 50          push         eax
```

Analysis

This vulnerability exists in Adobe Reader DC because it fails to parse the PDF file correctly. It's a use-after-free vulnerability based on the debug information in the previous section. Let's look into this specially crafted PDF file first. The comparison between the normal PDF file and the minimized PoC file is shown below.

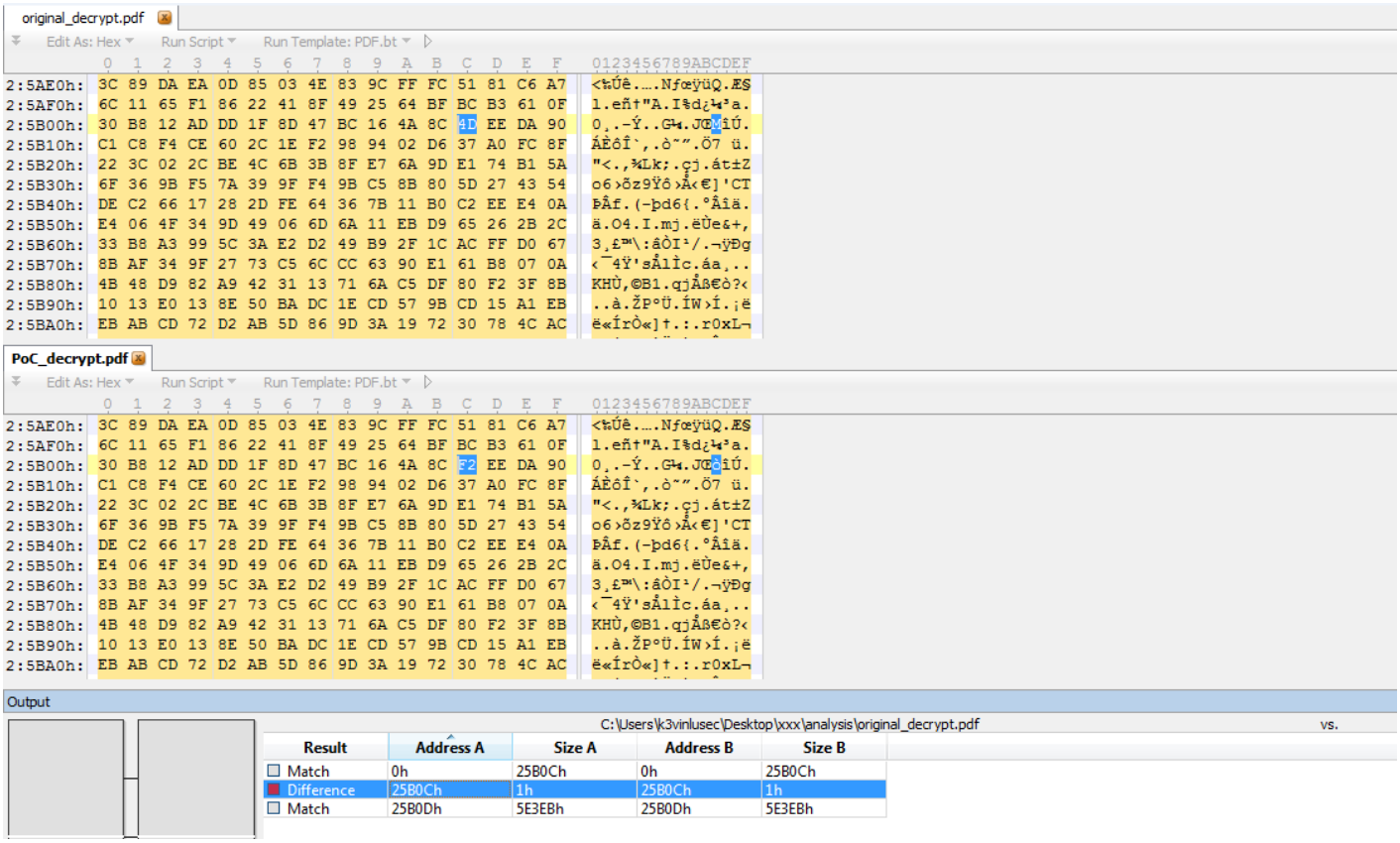

Figure 1. The PoC File vs The Original PDF File

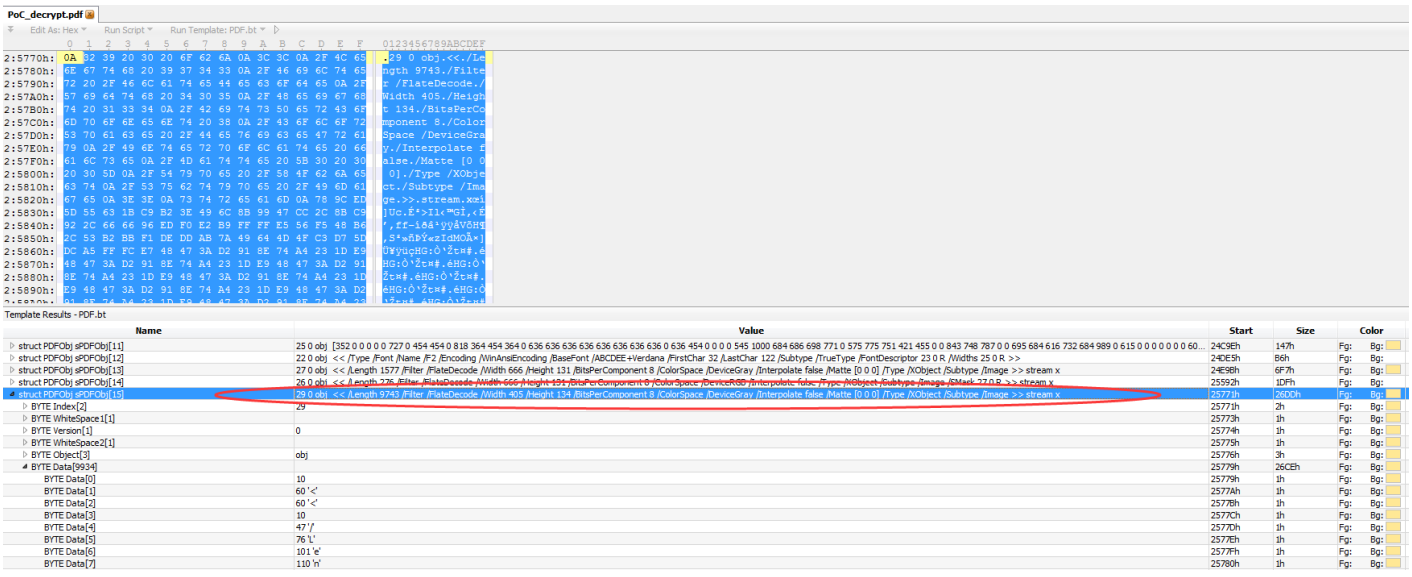

Figure 2. The Parsing of the PoC File with 010 Editor

In Figure 1 and Figure 2, the only difference is a single byte at offset 0x25B0C between the original PDF file and PoC file, and this byte is located in obj 29 in the PDF file. The object structure is shown below.

```
29 0 obj
<<
/Length 9743
/Filter /FlateDecode
/Width 405
/Height 134
/BitsPerComponent 8
/ColorSpace /DeviceGray
/Interpolate false
/Matte [0 0 0]
/Type /XObject
/Subtype /Image
>>
stream
..........
..........
..........
endstream
endobj
```

This object stores an image with width 405(0x195) and height 134(0x86). The data in it has been compressed using the deflate algorithm. (For more information on deflate, please refer to https://tools.ietf.org/html/rfc1951 (https://tools.ietf.org/html/rfc1951).) Zlib is a C library which implements the deflate algorithm. We can use a python script to decompress the data in obj 29. When the script is run, we can see the following exception.

```
0x2fc
0x2fd
0x2fe
0x2ff
0x300
0x301
0x302
0x303
Traceback (most recent call last):
  File "test.py", line 37, in <module>
    decompress(input,output)
  File "test.py", line 14, in decompress
    dst.write(decompress.decompress(data))
zlib.error: Error -3 while decompressing: invalid distance too far back
```

This means that some error occurs while decompressing the data. Only part of the data is successfully decompressed. The decompressed data is shown below and its length is 0x54FC.

```
5370h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5380h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5390h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53C0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53E0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
53F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5400h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5410h: 00 00 00 00 00 01 02 03 05 09 0E 13 1A 22 2A 32  ............."*2
5420h: 3A 41 47 4C 4F 53 55 56 56 56 55 54 52 4E 49 44  :AGLOSUVVVUTRNID
5430h: 3D 35 2C 24 1C 14 0E 09 05 03 01 01 02 04 07 0D  =5,$............
5440h: 14 1E 27 30 35 35 30 29 1F 16 0F 0A 0A 0D 13 1B  ..'0550)........
5450h: 25 2D 33 34 31 29 20 16 0E 08 04 02 01 00 00 00  %-341) .........
5460h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
5470h: 00 00 00 00 00 00 00 00 01 01 02 05 0A 11 1A 24 2D  ..............$-
5480h: 33 35 32 2C 22 19 10 0A 05 02 01 00 00 00 00 00  352,"...........
5490h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
54A0h: 00 00 00 00 00 00 00 00 00 00 00 01 01 03 07 0C 13  ...............
54B0h: 1D 27 30 36 38 35 2D 24 1A 12 06 08 0A 0D 61 20  .'0685-$......a
54C0h: 3E 00 00 00 00 00 00 00 00 00 00 00 00 44 00 00  >............D..
54D0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3C 17  ..............<.
54E0h: 00 00 00 31 18 00 17 34 55 00 00 00 68 58 49 00  ...1...4U...hXI.
54F0h: 00 00 00 00 00 00 00 5D 5E 3E 0B 50              .......]^>.P
```

Figure 3. Part of Decompressed Data

By reversing Adobe Reader DC, we can see that it also uses zlib to decompress the deflate-compressed data. So next, let's look at the crash information.

```
6021d2c6 8bf1           mov     esi,ecx
6021d2c8 8975f0         mov     dword ptr [ebp-10h],esi
6021d2cb c7064802d960   mov     dword ptr [esi],offset AcroRd32_60000000!CTJPEGThrowException+0x2586d8 (60d90248)
6021d2d1 8b8ec8000000   mov     ecx,dword ptr [esi+0C8h]
6021d2d7 33d2           xor     edx,edx
6021d2d9 42             inc     edx
6021d2da 8955fc         mov     dword ptr [ebp-4],edx
6021d2dd 85c9           test    ecx,ecx
6021d2df 7405           je      AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24666 (6021d2e6)
6021d2e1 8b01           mov     eax,dword ptr [ecx]  ds:002b:25b71fd8=????????    //crash here
6021d2e3 52             push    edx
6021d2e4 ff10           call    dword ptr [eax]
6021d2e6 83a6c800000000 and     dword ptr [esi+0C8h],0
6021d2ed 8b8618010000   mov     eax,dword ptr [esi+118h]
6021d2f3 85c0           test    eax,eax
6021d2f5 742a           je      AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x246a1 (6021d321)
1:027> r
eax=26ace5e0 ebx=07887004 ecx=25b71fd8 edx=00000001 esi=1cc81d14 edi=26bcd81c
eip=6021d2e1 esp=26ace5cc ebp=26ace5ec iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661:
6021d2e1 8b01           mov     eax,dword ptr [ecx]  ds:002b:25b71fd8=????????
1:027> dd esi+c8 L8
1cc81ddc  25b71fd8 445d95a6 00000052 00000000
1cc81dec  445d95a6 00000051 43e328cd 445d95a6
1:027> !heap -p -a esi
    address 1cc81d14 found in
    _DPH_HEAP_ROOT @ 2e21000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 1c9a2f08:         1cc811f0             fe0c -         1cc81000             11000
    11248e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    7df41d4e ntdll!RtlDebugAllocateHeap+0x00000030
    7defb586 ntdll!RtlpAllocateHeap+0x000000c4
    7dea3541 ntdll!RtlAllocateHeap+0x0000023a
    07f5ed63 MSVCR120!malloc+0x00000049
    07881f2d BIB!BIBInitialize4+0x000007ef
    07881d85 BIB!BIBInitialize4+0x00000647
    07884bb3 BIB!BIBInitialize4+0x00003475
    080950fc CoolType!CTInit+0x00000db1
    ....
```

From the above debug information, we can see that ECX points to an invalid memory address which is gotten from ESI+C8. ESI (address is 0x1cc81d14) is located in a heap buffer, with size 0xfe0c. So we need to figure out how the heap buffer is created, and how the data in it is handled. Following is the code snippet around the address 0x07881f2d.

```
07881f20 56              push    esi
07881f21 be0cfe0000      mov     esi,0FE0Ch
07881f26 56              push    esi
07881f27 ff156caa8907    call    dword ptr [BlBlBlBLockSmithUnlockImpl+0xed4b (0789aa6c)]
07881f2d 0135b4ab8907    add     dword ptr [BlBlBlBLockSmithUnlockImpl+0xee93 (0789abb4)],esi
07881f33 59              pop     ecx
07881f34 5e              pop     esi
07881f35 c3              ret
```

Let's set the following breakpoint in Windbg.

```
1:027> bu 07881f2d " .printf \"eax is: \\r\\"; dd eax; gc;"
1:027> bl
12 e 07881f2d     0001 (0001)  1:**** BlBlBlBlInitialize4+0x7ef " .printf \"eax is: \\r\\"; dd eax; gc;"
```

From the following debug information, we can see this breakpoint is hit several times.

```
…..
eax is:
1b05e1f0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e200  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e210  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e220  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e230  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e240  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e250  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1b05e260  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
eax is:
1a1e21f0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2200  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2210  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2220  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2230  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2240  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2250  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
1a1e2260  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
….
```

The next question, then, is how to find the heap buffer that causes the access violation. In Figure 3, we can see some ASCII characters in the decompressed data. We can search for them in memory and the results are shown below:

```
1:010> s -a 0x00000000 L?0xffffffff AGLOSUVVV
1a1e7619  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
1a1fe400  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
1b063619  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
```

There are three memory buffers that include the string. The debug information is shown below:

```
1:010> db 1a1e7619  L100
1a1e7619  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
1a1e7629  35 2c 24 1c 14 0e 09 05-03 01 01 02 04 07 0d 14  5,$.............
1a1e7639  1e 27 30 35 35 30 29 1f-16 0f 0a 0a 0d 13 1b 25  .'0550).........%
1a1e7649  2d 33 34 31 29 20 16 0e-08 04 02 01 00 00 00 00  -341) ..........
1a1e7659  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1a1e7669  00 00 00 00 00 00 01 01-02 05 0a 11 1a 24 2d 33  .............$-3
1a1e7679  35 32 2c 22 19 10 0a 05-02 01 00 00 00 00 00 00  52,"............
1a1e7689  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1a1e7699  00 00 00 00 00 00 00 00-00 01 01 03 07 0c 13 1d  ................
1a1e76a9  27 30 36 38 35 2d 24 1a-12 06 08 0a 0d 61 20 3e  '0685-$.......a >
1a1e76b9  00 00 00 00 00 00 00 00-00 00 00 00 44 00 00 00  ............D...
1a1e76c9  00 00 00 00 00 00 00 00-00 00 00 00 00 3c 17 00  .............<..
1a1e76d9  00 00 31 18 00 17 34 55-00 00 00 68 58 49 00 00  ..1...4U...hXI..
1a1e76e9  00 00 00 00 00 00 5d 5e-3e 0b 50 c0 c0 c0 c0 c0  ......]^>.P.....
1a1e76f9  c0 c0 c0 c0 c0 c0 c0 c0-c0 c0 c0 c0 c0 c0 c0 c0  ................
1a1e7709  c0 c0 c0 c0 c0 c0 c0 c0-c0 c0 c0 c0 c0 c0 c0 c0  ................
```

```
1:010> db 1a1fe400  L100
1a1fe400  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
1a1fe410  35 2c 24 1c 14 0e 09 05-03 01 01 02 04 07 0d 14  5,$.............
1a1fe420  1e 27 30 35 35 30 29 1f-16 0f 0a 0a 0d 13 1b 25  .'0550).........%
1a1fe430  2d 33 34 31 29 20 16 0e-08 04 02 01 00 00 00 00  -341) ..........
1a1fe440  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1a1fe450  00 00 00 00 00 00 01 01-02 05 0a 11 1a 24 2d 33  .............$-3
1a1fe460  35 32 2c 22 19 10 0a 05-02 01 00 00 00 00 00 00  52,"............
1a1fe470  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1a1fe480  00 00 00 00 00 00 00 00-00 01 01 03 07 0c 13 1d  ................
1a1fe490  27 30 36 38 35 2d 24 1a-12 06 08 0a 0d 61 20 3e  '0685-$.......a >
1a1fe4a0  00 00 00 00 00 00 00 00-00 00 00 00 44 00 00 00  ............D...
1a1fe4b0  00 00 00 00 00 00 00 00-00 00 00 00 00 3c 17 00  .............<..
1a1fe4c0  00 00 31 18 00 17 34 55-00 00 00 68 58 49 00 00  ..1...4U...hXI..
1a1fe4d0  00 00 00 00 00 00 5d 5e-3e 0b 50 00 00 00 00 00  ......]^>.P.....
1a1fe4e0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1a1fe4f0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

```
1:010> db 1b063619  L100
1b063619  41 47 4c 4f 53 55 56 56-56 55 54 52 4e 49 44 3d  AGLOSUVVVUTRNID=
1b063629  35 2c 24 1c 14 0e 09 05-03 01 01 02 04 07 0d 14  5,$.............
1b063639  1e 27 30 35 35 30 29 1f-16 0f 0a 0a 0d 13 1b 25  .'0550).......%
1b063649  2d 33 34 31 29 20 16 0e-08 04 02 01 00 00 00 00  -341) ..........
1b063659  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1b063669  00 00 00 00 00 00 01 01-02 05 0a 11 1a 24 2d 33  .............$-3
1b063679  35 32 2c 22 19 10 0a 05-02 01 00 00 00 00 00 00  52,"............
1b063689  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
1b063699  00 00 00 00 00 00 00 00-00 01 01 03 07 0c 13 1d  ................
1b0636a9  27 30 36 38 35 2d 24 1a-12 06 08 0a 0d 61 20 3e  '0685-$.....a >
1b0636b9  00 00 00 00 00 00 00 00-00 00 00 00 44 00 00 00  ............D...
1b0636c9  00 00 00 00 00 00 00 00-00 00 00 00 00 3c 17 00  .............<..
1b0636d9  00 00 31 18 00 17 34 55-00 00 00 68 58 49 00 00  ..1...4U...hXI..
1b0636e9  00 00 00 00 00 00 5d 5e-3e 0b 50 ff ff ff ff ff  ......]^>.P.....
1b0636f9  ff ff ff ff ff ff e6 ff-bc 6a 2e ff 04 00 00 00  .........j......
1b063709  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 ff  ................
```

```
1:010> !heap -p -a 1a1e7619
    address 1a1e7619 found in
    _DPH_HEAP_ROOT @ 47d1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                          1b02f1a0:         1a1e21f0             fe0c -         1a1e2000             11000
    11248e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    7df41d4e ntdll!RtlDebugAllocateHeap+0x00000030
    7defb586 ntdll!RtlpAllocateHeap+0x000000c4
    7dea3541 ntdll!RtlAllocateHeap+0x0000023a
    08bfed63 MSVCR120!malloc+0x00000049
    082d1f2d BIB!BIBInitialize4+0x000007ef
    082d1d85 BIB!BIBInitialize4+0x00000647
    082d5716 BIB!BIBInitialize4+0x00003fd8
    082d56bf BIB!BIBInitialize4+0x00003f81
    082d55aa BIB!BIBInitialize4+0x00003e6c
    ....
```

The above memory information matches the decompressed data in Figure 3. After repeating the debug, we found the memory address 0x1a1e7619 located in the heap buffer causes the access violation. The heap buffer starts at address 0x1a1e21f0, and its size is 0xfe0c. It contains the decompressed data whose size is 0x54fc. Next, we need to trace the heap buffer that starts at address 0x1a1e21f0. We can set the following breakpoint in Windbg:

```
1:010> bu 6056dc50
1:010> bl
12 e 081a1f2d     0001 (0001)  1:**** BIB!BIBInitialize4+0x7ef " .printf \"eax is: \\r\\n\"; dd eax; gc;"
13 e 6056dc50     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6e19
```

Continue to run until breakpoint 13 is hit on the 4th time. Following is the debug information:

```
1:010> g
....
1:026> g
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
Breakpoint 13 hit   // hit on the 4th time
eax=1a7ae218 ebx=1a7ae4c8 ecx=1a1f10bc edx=60d902ac esi=00000000 edi=1a7ae414
eip=6056dc50 esp=1a7ae1e0 ebp=1a7ae248 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6e19:
6056dc50 68ec000000      push    0ECh
1:026> !heap -p -a ecx
    address 1a1f10bc found in
    _DPH_HEAP_ROOT @ 47d1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                1b02f1a0:         1a1e21f0             fe0c -         1a1e2000             11000
    11248e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    7df41d4e ntdll!RtlDebugAllocateHeap+0x00000030
    7defb586 ntdll!RtlpAllocateHeap+0x000000c4
    7dea3541 ntdll!RtlAllocateHeap+0x0000023a
    08bfed63 MSVCR120!malloc+0x00000049
    082d1f2d BIB!BIBInitialize4+0x000007ef
    082d1d85 BIB!BIBInitialize4+0x00000647
    082d5716 BIB!BIBInitialize4+0x00003fd8
    082d56bf BIB!BIBInitialize4+0x00003f81
    082d55aa BIB!BIBInitialize4+0x00003e6c
    ....

1:026> dd ecx
1a1f10bc  60d902ac 1a1ef500 00000000 00010000
1a1f10cc  00000000 00000000 00000000 00000000
1a1f10dc  197a2fe0 ffffffff 00000000 00000000
1a1f10ec  00000000 00000000 1a1f0c6c 1a1f0db8
1a1f10fc  00000086 00000195 00000008 00000004
1a1f110c  00000654 00000008 00000008 00000000
1a1f111c  00000000 c0c0c000 00ac0068 18f93d9c
1a1f112c  00000000 3f800000 00000000 00000000
```

```
1:026> dd poi(ecx+38)
1a1f0c6c  60d90248 1a1f1200 00000000 00010000
1a1f0c7c  00000000 00000000 00000000 00000000
1a1f0c8c  1b73dfe0 ffffffff 00000000 00000000
1a1f0c9c  00000000 00000000 00000195 00000086
1a1f0cac  00000008 00000008 00000003 00000000
1a1f0cbc  00027bfa 00000000 000004bf c0000000
1a1f0ccc  0000001c 00000000 00000000 bf800000
1a1f0cdc  bf800000 00000001 00000001 00000000
```

```
1:026> dd poi(ecx+3c)
1a1f0db8  60d90248 00000000 00000000 00010000
1a1f0dc8  00000000 00000000 00000000 00000000
1a1f0dd8  16fdafe0 ffffffff 00000000 00000000
1a1f0de8  00000000 00000000 00000195 00000086
1a1f0df8  00000008 00000008 00000001 00000000
1a1f0e08  0000d3fe 00000000 00000195 c0000000
1a1f0e18  0000001d 00000000 00000000 bf800000ss
1a1f0e28  bf800000 00000001 00000001 00000000
```

```
1:026> ba w4 1a1f0c6c+c8       //set memory write breakpoint
1:026> ba w4 1a1f0db8+c8       //set memory write breakpoint
```

When we continue to run, we get the following debug information:

```
601ee433 7408            je      AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144eec (601ee43d)
601ee435 898ec8000000    mov     dword ptr [esi+0C8h],ecx  //trigger the memory write breakpoint
601ee43b eb51            jmp     AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144f3d (601ee48e)
601ee43d 8b01            mov     eax,dword ptr [ecx]
601ee43f 6a01            push    1
601ee441 ff10            call    dword ptr [eax]
```

```
Breakpoint 14 hit
eax=1940dfd8 ebx=1940dfd8 ecx=1940dfd8 edx=1a7ae068 esi=1a1f0c6c edi=00000000
eip=601ee43b esp=1a7ae0a8 ebp=1a7ae0cc iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144eea:
601ee43b eb51            jmp     AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144f3d (601ee48e)
```

```
1:026> bl
12 e 082d1f2d     0001 (0001)  1:**** BlBlBlBlInitialize4+0x7ef " .printf \"eax is: \\n\"; dd eax; gc;"
13 e 6056dc50     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6e19
14 e 1a1f0d34 w 4 0001 (0001)  1:****
15 e 1a1f0e80 w 4 0001 (0001)  1:****
```

```
1:026> dd ecx L16
1940dfd8  60da2120 1a1f0ca4 60d8ebec 18b5ffe0
1940dfe8  1bc90ff8 00000000 60d900e4 00000000
1940dff8  00000000 d0d0d0d0 ???????? ????????
1940e008  ???????? ???????? ???????? ????????
```

```
1:026>  !heap -p -a ecx
    address 1940dfd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 18a90b2c:         1940dfd8               24 -         1940d000             2000
          ? AcroRd32_60000000!CTJPEGThrowException+26a5b0
    11248e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    7df41d4e ntdll!RtlDebugAllocateHeap+0x00000030
    7defb586 ntdll!RtlpAllocateHeap+0x000000c4
    7dea3541 ntdll!RtlAllocateHeap+0x0000023a
    08c011f9 MSVCR120!_calloc_impl+0x00000045
    08c0cc17 MSVCR120!calloc+0x00000018
    60049926 AcroRd32_60000000!AcroWinBrowserMain+0x00001fd6
    601ee3f4 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x00144ea3
    6056ddbb AcroRd32_60000000!AX_PDXlateToHostEx+0x001f6f84
    601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
    086ad5ff AGM!AGMInitialize+0x00054da3
```

```
1:026> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
1a7ae0cc 6056ddbb 95f09b6d 1a7ae414 00000000 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144eea
1a7ae1dc 601ec89f 1a7ae218 1a7ae234 00000000 AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6f84
1a7ae248 086ad5ff 1a1f10bc 1a7ae414 00000000 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x14334e
1a7ae2f4 082d226e 082d4b2e 082d4b3e 95f099a6 AGM!AGMInitialize+0x54da3
....
6056ddb6 e82206c8ff      call    AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144e8c (601ee3dd)//trigger the memory write breakpoint 14 in sub_601ee3dd.
6056ddbb 8b4e3c          mov     ecx,dword ptr [esi+3Ch]
```

```
1:026> g
Breakpoint 15 hit
eax=1b8f4fd8 ebx=1b8f4fd8 ecx=1b8f4fd8 edx=1a7ae068 esi=1a1f0db8 edi=00000000
eip=601ee43b esp=1a7ae0a8 ebp=1a7ae0cc iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144eea:
601ee43b eb51            jmp     AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144f3d (601ee48e)
```

```
1:026> dd ecx L16
1b8f4fd8  60da2120 1a1f0df0 60d8ebec 1b902fe0
1b8f4fe8  1271eff8 00000000 60d900e4 00000000
1b8f4ff8  00000000 d0d0d0d0 ???????? ????????
1b8f5008  ???????? ???????? ???????? ????????
```

```
1:026> !heap -p -a ecx
    address 1b8f4fd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                 1bbd1f70:         1b8f4fd8               24 -         1b8f4000             2000
          ? AcroRd32_60000000!CTJPEGThrowException+26a5b0
    11248e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    7df41d4e ntdll!RtlDebugAllocateHeap+0x00000030
    7defb586 ntdll!RtlpAllocateHeap+0x000000c4
    7dea3541 ntdll!RtlAllocateHeap+0x0000023a
    08c011f9 MSVCR120!_calloc_impl+0x00000045
    08c0cc17 MSVCR120!calloc+0x00000018
    60049926 AcroRd32_60000000!AcroWinBrowserMain+0x00001fd6
    601ee3f4 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x00144ea3
    6056de94 AcroRd32_60000000!AX_PDXlateToHostEx+0x001f705d
    601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
    086ad5ff AGM!AGMInitialize+0x00054da3

1:026> kb
ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
1a7ae0cc 6056de94 95f09b6d 1a7ae414 00000000 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144eea
1a7ae1dc 601ec89f 1a7ae218 1a7ae234 00000000 AcroRd32_60000000!AX_PDXlateToHostEx+0x1f705d
1a7ae248 086ad5ff 1a1f10bc 1a7ae414 00000000 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x14334e
1a7ae2f4 082d226e 082d4b2e 082d4b3e 95f099a6 AGM!AGMInitialize+0x54da3
1a7ae2f8 082d4b2e 082d4b3e 95f099a6 1a7ae400 BIB!BIBInitialize4+0xb30
1a7ae2fc 082d4b3e 95f099a6 1a7ae400 1a7ae34c BIB!BIBInitialize4+0x33f0
1a7ae330 08681921 08681929 95f09903 95f09893 BIB!BIBInitialize4+0x3400
1a7ae390 086ad389 16e47d90 00000000 1a7ae400 AGM!AGMInitialize+0x290c5
....
```

```
6056de8d eb05          jmp     AcroRd32_60000000!AX_PDXlateToHostEx+0x1f705d (6056de94)
6056de8f e84905c8ff    call    AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x144e8c (601ee3dd)  //trigger the memory write breakpoint 15 in sub_601ee3dd.
6056de94 8b75d0        mov     esi,dword ptr [ebp-30h]s
```

Following is the 'if' branch in function sub_6056dc50. Because [ebp+var_8C] is equal to 0, it will take the green color branch. In this branch, the pointer of the heap buffer isn't set to NULL after the heap buffer is freed. So this is what causes the use-after-free issue.

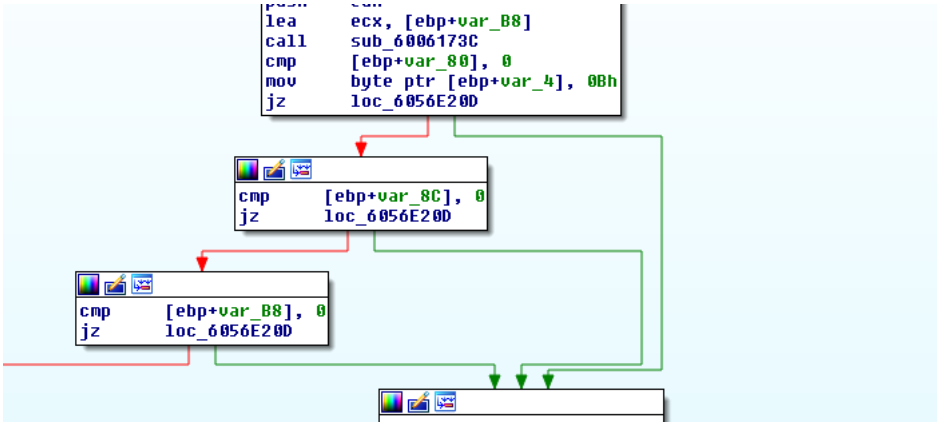

Figure 4. 'If' Branch in Function sub_6056dc50

Let's set the following breakpoint to trace why [ebp-8c] is equal to zero:

```
1:026> bu 6056DF22
1:026> bl
12 e 08011f2d     0001 (0001)  1:**** BlBlBlBlnitialize4+0x7ef " .printf \"eax is: \\r\\"; dd eax; gc;"
13 e 6056dc50     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6e19
14 e 1c85e878 w 4 0001 (0001)  1:****
15 e 1c85e9c4 w 4 0001 (0001)  1:****
16 e 6056df22     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f70eb
```

Continue to run:

```
1:026> g
Breakpoint 16 hit
eax=60da2120 ebx=1a1f10dc ecx=1b8f4fd8 edx=1a7ae13c esi=00000000 edi=1a1f10bc
eip=6056df22 esp=1a7ae0c0 ebp=1a7ae1dc iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!AX_PDXlateToHostEx+0x1f70eb:
6056df22 ff5004          call    dword ptr [eax+4]    ds:002b:60da2124=601f6ee9
1:026> dds esp L5
1a7ae0c0  00000000
1a7ae0c4  00000086
1a7ae0c8  1a7ae13c
1a7ae0cc  1a7ae134
1a7ae0d0  1a7ae150    //ebp-8c
1:026> dd 1a7ae150 L1
1a7ae150  00000000
```

Following is the C code of function sub_601F6EE9:

```
unsigned int *__thiscall sub_601F6EE9(int this, int a2, unsigned int a3, _DWORD *a4, unsigned int *a5, int a6)
{
  ……
  v6 = this;
  v7 = _RTDynamicCast(*(_DWORD *)(*(_DWORD *)(this + 12) + 28), 0, &off_61276C18, &off_61276CBC, 0);
  v8 = v7;
  if ( !v7 )
  {
    sub_60177BFC(537329694, 0);
    a3 = 0;
    goto LABEL_3;
  }
  v9 = *(_DWORD *)(v7 + 36);    //v9 is 0
  if ( v9 < a3 )           //a3 is the height 0x86
  {  //take this branch
    v10 = *(_DWORD *)(*(_DWORD *)(v6 + 4) + 32);    //v10 is the width 0x195
    v11 = v10 * (a3 - v9);    // v11 = 0x86*0x195 = 0xd3fe
    v12 = v9 * v10;
    v13 = *(_DWORD *)(v6 + 28);    //v13 is 0
    v31 = v11;     // v31 is 0xd3fe
    v14 = *(_DWORD *)(v7 + 8) + v12;
    v28 = v14;
    if ( v13 )
    {
      v15 = (*(int (__thiscall **)(int, int))(*(_DWORD *)v13 + 12))(v13, v11);
      v32 = v15;
      v16 = sub_600511EC(v15);
      v17 = *(_DWORD *)(v8 + 32);
      v18 = v16;
      v29 = v16;
      v30 = sub_60107100(v16, 1, v32, v17);
      if ( v30 != v32 )
      {
        if ( v18 )
          sub_60049EE7();
        goto LABEL_9;
      }
      v19 = *(_DWORD *)(v6 + 4);
      v20 = *(_DWORD *)v19;
      v31 = v30 / (v32 / v31) / *(_DWORD *)(v19 + 32);
      v30 = v31 * v20;
      if ( v29 )
        (*(void (__stdcall **)(int, unsigned int, unsigned int, _DWORD))(**(_DWORD **)(v6 + 28) + 4))(v29, v28, v30, 0);
      sub_60049EE7();
      *(_DWORD *)(v8 + 36) += v31;
    }
    else
    {  //take this branch
      v21 = sub_60107100(v14, 1, v11, *(_DWORD *)(v7 + 32)); //v21=0x54fc, it's the length of decompressed data. Trace it.
      v28 = v21;
      if ( v21 != v31 ) //v31 = 0xd3fe,v21=0x54fc, so this condition is true.
      { //take this branch
LABEL_9:
        sub_60177BFC(537329694, 0);
        a3 = 0;
LABEL_3:
        CxxThrowException(&a3, &unk_610BAC3C);    // throw an exception, go to the end of function.
```

```
      }
      *(_DWORD *)(v8 + 36) += v21 / *(_DWORD *)(*(_DWORD *)(v6 + 4) + 32);
      if ( *(_DWORD *)(*(_DWORD *)(v6 + 4) + 8) == 16 )
        sub_60592904(v14, v28 >> 1);
    }
    if ( *(_DWORD *)(v8 + 36) == *(_DWORD *)(*(_DWORD *)(v6 + 4) + 4) )
    {
      sub_600620E0(*(_DWORD *)(v8 + 32));
      *(_DWORD *)(v8 + 32) = 0;
    }
  }
  v22 = a4;
  *a4 = a2;
  v23 = *(_DWORD *)(v8 + 36);
  result = a5;
  if ( a3 < v23 )
    v23 = a3;
  *a5 = v23;
  if ( v23 > *v22 )
  {
    v25 = *(_DWORD *)(*(_DWORD *)(v6 + 4) + 32) * (v23 - *v22);
    v26 = sub_6006173C(&v27, v25, 0);
    v33 = 0;
    sub_600618E5((_DWORD *)a6, (int)v26);  //in function sub_600618E5, it will assign a non-NULL to pointer a6. This branch is not taken. So a6 is still NULL.
    v33 = -1;
    sub_6006169C(&v27);
    if ( !*(_DWORD *)a6 )s
    {
      sub_60177BFC(1073741826, 0);
      a6 = 0;
      CxxThrowException(&a6, &unk_610BAC3C);
    }
    result = (unsigned int *)sub_6004E7CC(
                               *(void **)(a6 + 4),
                               (void *)(*(_DWORD *)(v8 + 8) + *a4 * *(_DWORD *)(*(_DWORD *)(v6 + 4) + 32)),
                               v25);
  }
  return result;
}
```

Next, enter the function sub_60107100.

```
60107100 55              push    ebp
60107101 8bec            mov     ebp,esp
60107103 8b4514          mov     eax,dword ptr [ebp+14h]
60107106 50              push    eax
60107107 ff7510          push    dword ptr [ebp+10h]
6010710a ff750c          push    dword ptr [ebp+0Ch]
6010710d 8b4810          mov     ecx,dword ptr [eax+10h]
60107110 ff7508          push    dword ptr [ebp+8]
60107113 ff5108          call    dword ptr [ecx+8]               ds:002b:60db8794=60061927  // trace it.
60107116 83c410          add     esp,10h
60107119 5d              pop     ebp
6010711a c3              ret
```

```
1:026> t
eax=193e9fb8 ebx=1b8f4fd8 ecx=60db878c edx=00000000 esi=1a1e21f8 edi=19403fd8
eip=60107113 esp=1a7ae054 ebp=1a7ae064 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x5dbc2:
60107113 ff5108          call    dword ptr [ecx+8]    ds:002b:60db8794=60061927
1:026> dds 1a7ae054 L4
1a7ae054  1a1e21f8
1a7ae058  00000001
1a7ae05c  0000d3fe
1a7ae060  193e9fb8
```

Then enter the function sub_60061927:

```
int __cdecl sub_60061927(int a1, signed int a2, int a3, int a4)
{
  void *v4; // ecx@2
  if ( a4 )
    v4 = (void *)(a4 - 8);
  else
    v4 = 0;
  return sub_60061948(v4, a1, a2, a3);    //trace it.
}
```

Then enter the function sub_60061948:

```
int __thiscall sub_60061948(void *this, int a2, signed int a3, int a4)      // get the length of decompressed data
{
  void *v4; // esi@1
  int v5; // ebx@3
  int result; // eax@4
  char v7; // [sp+Ch] [bp-8h]@5
  v4 = this;
  if ( a3 && a4 > 0x7FFFFFFF / a3 )
  {
    sub_60177BFC(1074397191, 0);
    a4 = 0;
    CxxThrowException(&a4, &unk_610BAC3C);
  }
  v5 = a4 * a3;
  if ( a4 * a3 >= 1 )
  {
    sub_60060BAB((int)this, 0, 1074397190);
    (*(void (__thiscall **)(void *, char *, int, int))(*(_DWORD *)v4 + 8))(v4, &v7, a2, v5); //
    sub_600603CE((int)&v7);
    if ( (*((_DWORD *)v4 + 10) >> 1) & 1 )
      *((_DWORD *)v4 + 5) |= 8u;
    if ( *((_BYTE *)v4 + 40) & 5 )
      *((_DWORD *)v4 + 5) |= 0x10u;
    result = *((_DWORD *)v4 + 13);     // here the result is 0x54fc.
    if ( a3 != 1 )
      result = (a3 + result - 1) / a3;
  }
  else
  {
    result = 0;
  }
  return result;     //return 0x54fc.
}
```

From the above analysis, we know why [ebp-8c] is equal to 0. Next, let's set the following breakpoint to take the "if" branch we saw in Figure 4.

```
1:026> bu 6056dfed
1:026> bl
12 e 082d1f2d     0001 (0001)  1:**** BIBIBIBInitialize4+0x7ef " .printf \"eax is: \\r\\"; dd eax; gc;"
13 e 6056dc50     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f6e19
14 e 1a1f0d34 w 4 0001 (0001)  1:****
15 e 1a1f0e80 w 4 0001 (0001)  1:****
16 e 6056df22     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f70eb
17 e 6056dfed     0001 (0001)  1:**** AcroRd32_60000000!AX_PDXlateToHostEx+0x1f71b6
```

Continue to run. We can see the following debug information:

```
1:026> g
(2ad4.313c): C++ EH exception - code e06d7363 (first chance)
Breakpoint 17 hit
eax=1a7ae124 ebx=1a1f10dc ecx=1b047cf8 edx=6138d01c esi=00000001 edi=1a1f10bc
eip=6056dfed esp=1a7ae0d4 ebp=1a7ae1dc iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!AX_PDXlateToHostEx+0x1f71b6:
6056dfed 837d8000        cmp     dword ptr [ebp-80h],0 ss:002b:1a7ae15c=1b047da0
1:026> dd ebp-80 L1
1a7ae15c  1b047da0
1:026> dd ebp-8c L1
1a7ae150  00000000
1:026> dd ebp-b8 L1
1a7ae124  1b047cf8
```

Because [ebp-8c] is equal to 0, the program takes the green color branch. If it takes the red color branch, the heap buffer pointed by [this+0x38]+0xC8 will be freed and set [[this+0x38]+0xC8] to NULL.

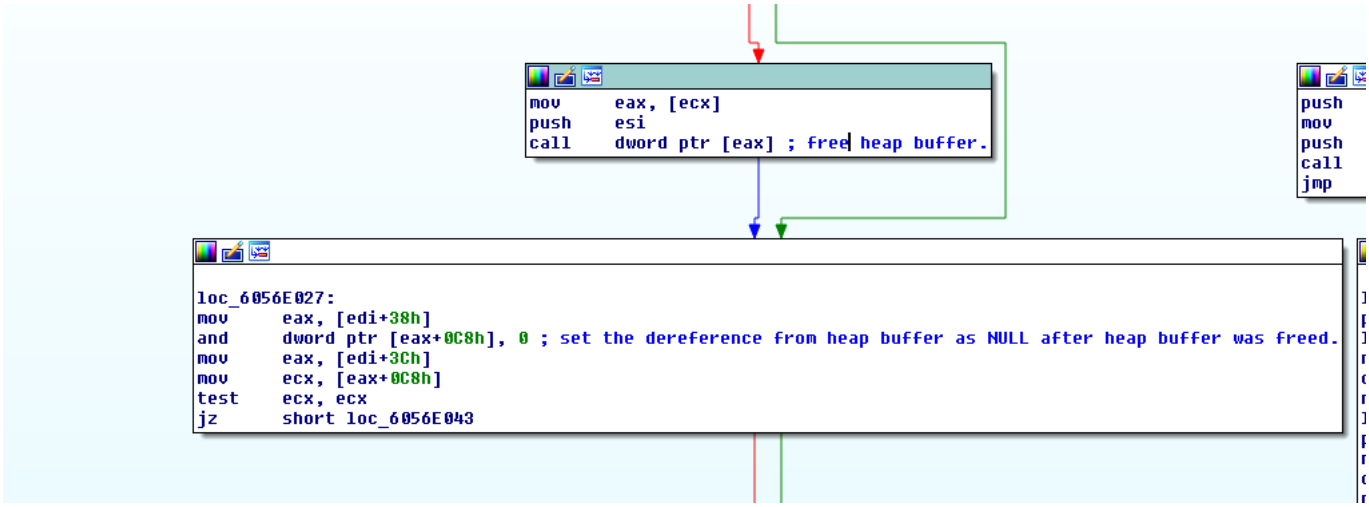

Figure 5. Red Color Branch 1

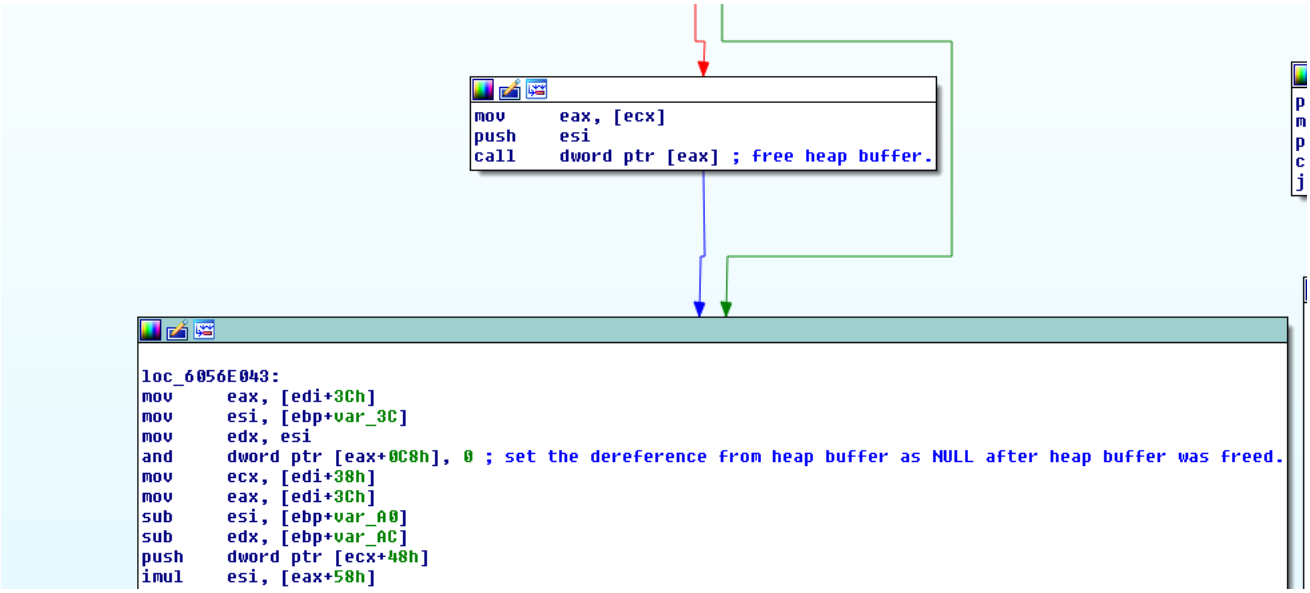

Figure 6. Red Color Branch 2

Take the green color branch as follows.

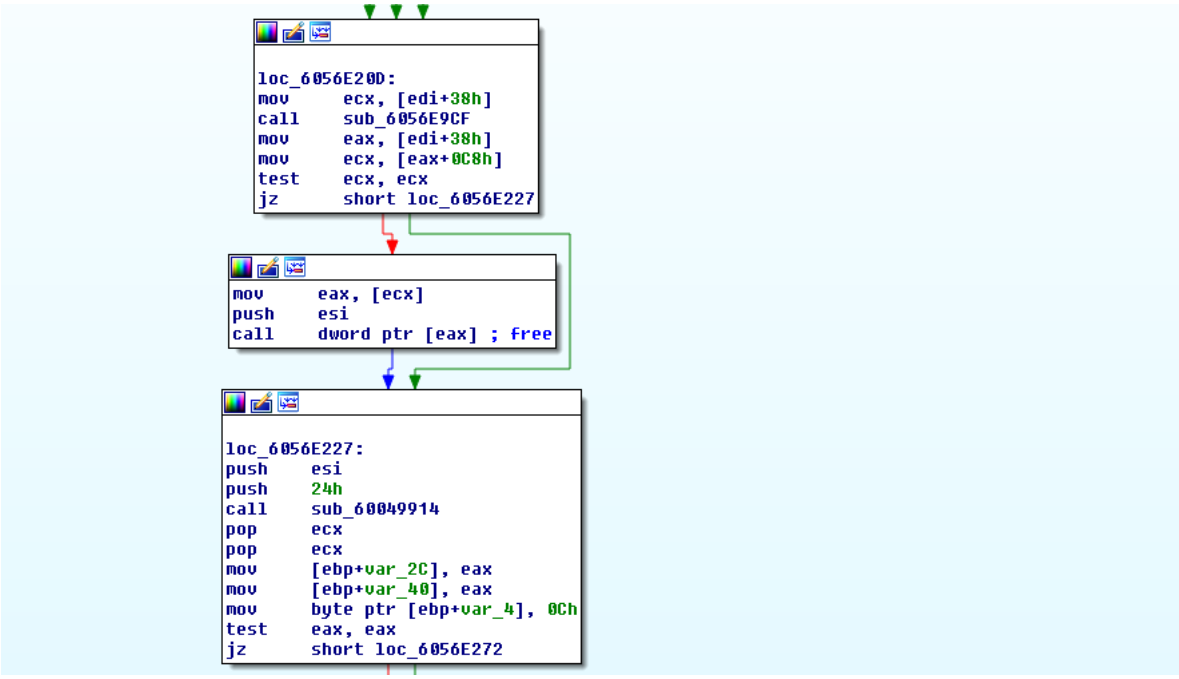

Figure 7. Green Color Branch

```
1:026> t
eax=1a7ae124 ebx=1a1f10dc ecx=1b047cf8 edx=6138d01c esi=00000001 edi=1a1f10bc
eip=6056e20d esp=1a7ae0d4 ebp=1a7ae1dc iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
AcroRd32_60000000!AX_PDXlateToHostEx+0x1f73d6:
6056e20d 8b4f38          mov     ecx,dword ptr [edi+38h] ds:002b:1a1f10f4=1a1f0c6c
1:026> dd 1a1f10bc
1a1f10bc  60d902ac 1a1ef500 2007001e 00010000
1a1f10cc  00000000 00000000 00000000 00000000
1a1f10dc  197a2fe0 fffffffe 00000001 0000313c
1a1f10ec  00000000 00000000 1a1f0c6c 1a1f0db8
1a1f10fc  00000086 00000195 00000008 00000004
1a1f110c  00000654 00000008 00000008 00000000
1a1f111c  00000000 c0c0c000 00ac0068 18f93d9c
1a1f112c  00000000 3f800000 00000000 00000000
1:026> dd 1a1f0c6c+c8 L4
1a1f0d34  1940dfd8 444fe4e8 43b6b2e8 00000000
1:026> dd 1a1f0db8+c8 L4
1a1f0e80  1b8f4fd8 445d95a6 00000052 00000000
1:026> dd 1940dfd8 L16
1940dfd8  60da2120 1a1f0ca4 60d8ebec 18b5ffe0
1940dfe8  1bc90ff8 00000000 60d900e4 00000000
1940dff8  00000000 d0d0d0d0 ???????? ????????
1940e008  ???????? ???????? ???????? ????????
1:026> dd 1b8f4fd8 L16
1b8f4fd8  60da2120 1a1f0df0 60d8ebec 1b902fe0
1b8f4fe8  1271eff8 00000000 60d900e4 00000000
1b8f4ff8  00000000 d0d0d0d0 ???????? ????????
1b8f5008  ???????? ???????? ???????? ????????
```

Set the following breakpoint to check when the heap buffer 0x1940dfd8 and 0x1b8f4fd8 are freed:

```
bu AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b62
```

Continue to run.

```
Breakpoint 20 hit
eax=1bc90ff8 ebx=1a1f10dc ecx=601fe90e edx=00000001 esi=1940dfd8 edi=1a1f10bc
eip=601fe7e2 esp=1a7ae0c4 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b62:
601fe7e2 56              push    esi
```

```
1:026> t
eax=1bc90ff8 ebx=1a1f10dc ecx=601fe90e edx=00000001 esi=1940dfd8 edi=1a1f10bc
eip=601fe7e3 esp=1a7ae0c0 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b63:
601fe7e3 e8ffb6e4ff      call    AcroRd32_60000000!AcroWinBrowserMain+0x2597 (60049ee7)
1:026> dd esi L16
1940dfd8  60d1f95c 1a1f0ca4 60d1f95c 18b5ffe0
1940dfe8  1bc90ff8 00000000 60d1f95c 00000000
1940dff8  00000000 d0d0d0d0 ???????? ????????
1940e008  ???????? ???????? ???????? ????????
```

```
1:026> p
eax=00000001 ebx=1a1f10dc ecx=7dea30ed edx=047d1078 esi=1940dfd8 edi=1a1f10bc
eip=601fe7e8 esp=1a7ae0c0 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b68:
601fe7e8 59              pop     ecx
1:026> dd esi L16
1940dfd8  ???????? ???????? ???????? ????????
1940dfe8  ???????? ???????? ???????? ????????
1940dff8  ???????? ???????? ???????? ????????
1940e008  ???????? ???????? ???????? ????????
```

```
1:026> !heap -p -a esi
    address 1940dfd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                 18a90b2c:         1940d000             2000
    112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
    7df4251c ntdll!RtlDebugFreeHeap+0x0000002f
    7defb2a2 ntdll!RtlpFreeHeap+0x0000005d
    7dea2ce5 ntdll!RtlFreeHeap+0x00000142
    7dd714bd kernel32!HeapFree+0x00000014
    08bfecfa MSVCR120!free+0x0000001a
    601fe7e8 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00005b68
    6056e227 AcroRd32_60000000!AX_PDXlateToHostEx+0x001f73f0
    601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
    086ad5ff AGM!AGMInitialize+0x00054da3
```

We can now see that the heap buffer 0x1940dfd8 has been freed. Continue to run.

```
1:026> g
Breakpoint 14 hit
eax=1a1f0c6c ebx=1a1f10dc ecx=1bc7afd8 edx=6138d01c esi=00000001 edi=1a1f10bc
eip=6056e281 esp=1a7ae0d4 ebp=1a7ae1dc iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
AcroRd32_60000000!AX_PDXlateToHostEx+0x1f744a:
6056e281 8b4f3c          mov     ecx,dword ptr [edi+3Ch] ds:002b:1a1f10f8=1a1f0db8
```

```
1:026> dd 1a1f0d34 L4
1a1f0d34  1bc7afd8 444fe4e8 43b6b2e8 00000000
1:026> dd ecx L16
1bc7afd8  60da2110 1a1f0ca4 60d8ebec 1bc6afe0
1bc7afe8  1a559ff8 00000000 60d900e4 00000000
1bc7aff8  00000000 d0d0d0d0 ???????? ????????
1bc7b008  ???????? ???????? ???????? ????????
```

```
1:026> g
Breakpoint 20 hit
eax=1271eff8 ebx=1a1f10dc ecx=601fe90e edx=00000001 esi=1b8f4fd8 edi=1a1f10bc
eip=601fe7e2 esp=1a7ae0c4 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b62:
601fe7e2 56              push    esi
```

```
1:026> p
eax=1271eff8 ebx=1a1f10dc ecx=601fe90e edx=00000001 esi=1b8f4fd8 edi=1a1f10bc
eip=601fe7e3 esp=1a7ae0c0 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b63:
601fe7e3 e8ffb6e4ff      call    AcroRd32_60000000!AcroWinBrowserMain+0x2597 (60049ee7)
```

```
1:026> dd 1b8f4fd8 L16
1b8f4fd8  60d1f95c 1a1f0df0 60d1f95c 1b902fe0
1b8f4fe8  1271eff8 00000000 60d1f95c 00000000
1b8f4ff8  00000000 d0d0d0d0 ???????? ????????
1b8f5008  ???????? ???????? ???????? ????????
```

```
1:026> p
eax=00000001 ebx=1a1f10dc ecx=7dea30ed edx=047d1078 esi=1b8f4fd8 edi=1a1f10bc
eip=601fe7e8 esp=1a7ae0c0 ebp=1a7ae0c8 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x5b68:
601fe7e8 59              pop     ecx
```

```
1:026> !heap -p -a esi
    address 1b8f4fd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                  1bbd1f70:         1b8f4000             2000
    112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
    7df4251c ntdll!RtlDebugFreeHeap+0x0000002f
    7defb2a2 ntdll!RtlpFreeHeap+0x0000005d
    7dea2ce5 ntdll!RtlFreeHeap+0x00000142
    7dd714bd kernel32!HeapFree+0x00000014
    08bfecfa MSVCR120!free+0x0000001a
    601fe7e8 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00005b68
    6056e29b AcroRd32_60000000!AX_PDXlateToHostEx+0x001f7464
    601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
    086ad5ff AGM!AGMInitialize+0x00054da3
```

We can see the data at address 0x1a1f0d34 is set to 0x1bc7afd8 and the heap buffer 0x1b8f4fd8 has been freed. Continue to run.

```
Breakpoint 14 hit
eax=1bc7afd8 ebx=082d7004 ecx=1bc7afd8 edx=047d1078 esi=1a1f0c6c edi=1b048c40
eip=6021d2ed esp=1a7ae5cc ebp=1a7ae5ec iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x2466d:
6021d2ed 8b8618010000    mov     eax,dword ptr [esi+118h] ds:002b:1a1f0d84=00000000
```

```
1:026> dd esi+c8 L4
1a1f0d34  00000000 444fe4e8 43b6b2e8 00000000
1:026> dd 1bc7afd8 L16
1bc7afd8  ???????? ???????? ???????? ????????
1bc7afe8  ???????? ???????? ???????? ????????
1bc7aff8  ???????? ???????? ???????? ????????
1bc7b008  ???????? ???????? ???????? ????????
```

```
1:026> !heap -p -a 1bc7afd8
    address 1bc7afd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                  1bbd10d0:         1bc7a000             2000
    112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
    7df4251c ntdll!RtlDebugFreeHeap+0x0000002f
    7defb2a2 ntdll!RtlpFreeHeap+0x0000005d
    7dea2ce5 ntdll!RtlFreeHeap+0x00000142
    7dd714bd kernel32!HeapFree+0x00000014
    08bfecfa MSVCR120!free+0x0000001a
    60592484 AcroRd32_60000000!AX_PDXlateToHostEx+0x0021b64d
    6021d2e6 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00024666
    6021d2a3 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00024623
    6056bd24 AcroRd32_60000000!AX_PDXlateToHostEx+0x001f4eed
    6056bf9f AcroRd32_60000000!AX_PDXlateToHostEx+0x001f5168
    601253d0 AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0007be7f
    086be13b AGM!AGMInitialize+0x000658df
```

We can see the heap buffer 0x1bc7afd8 has been freed and the data at address 0x1a1f0d34 is set to NULL. Continue to run. This is where the crash occurs.

```
1:026> g
(2ad4.313c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=1a7ae5e0 ebx=082d7004 ecx=1b8f4fd8 edx=00000001 esi=1a1f0db8 edi=1b048c40
eip=6021d2e1 esp=1a7ae5cc ebp=1a7ae5ec iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661:
6021d2e1 8b01            mov     eax,dword ptr [ecx]  ds:002b:1b8f4fd8=????????
```

```
1:026> !heap -p -a ecx
    address 1b8f4fd8 found in
    _DPH_HEAP_ROOT @ 47d1000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                  1bbd1f70:         1b8f4000             2000
    112490b2 verifier!AVrfDebugPageHeapFree+0x000000c2
    7df4251c ntdll!RtlDebugFreeHeap+0x0000002f
    7defb2a2 ntdll!RtlpFreeHeap+0x0000005d
    7dea2ce5 ntdll!RtlFreeHeap+0x00000142
    7dd714bd kernel32!HeapFree+0x00000014
    08bfecfa MSVCR120!free+0x0000001a
    601fe7e8 AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x00005b68
    6056e29b AcroRd32_60000000!AX_PDXlateToHostEx+0x001f7464
    601ec89f AcroRd32_60000000!CTJPEGWriter::CTJPEGWriter+0x0014334e
    086ad5ff AGM!AGMInitialize+0x00054da3
```

```
1:026> dd esi+c8 L4
1a1f0e80  1b8f4fd8 445d95a6 00000052 00000000
```

```
1:026> u
AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x24661:
6021d2e1 8b01            mov     eax,dword ptr [ecx]   //crash here. Eax is vtable pointer.
6021d2e3 52              push    edx
6021d2e4 ff10            call    dword ptr [eax]       //call the function in vtable.
6021d2e6 83a6c800000000  and     dword ptr [esi+0C8h],0
6021d2ed 8b8618010000    mov     eax,dword ptr [esi+118h]
6021d2f3 85c0            test    eax,eax
6021d2f5 742a            je      AcroRd32_60000000!CTJPEGDecoderReadNextTile+0x246a1 (6021d321)
6021d2f7 50              push    eax
```

Based on the above analysis, we can finally draw a picture to explain how the heap buffer is created, freed, and then reused.

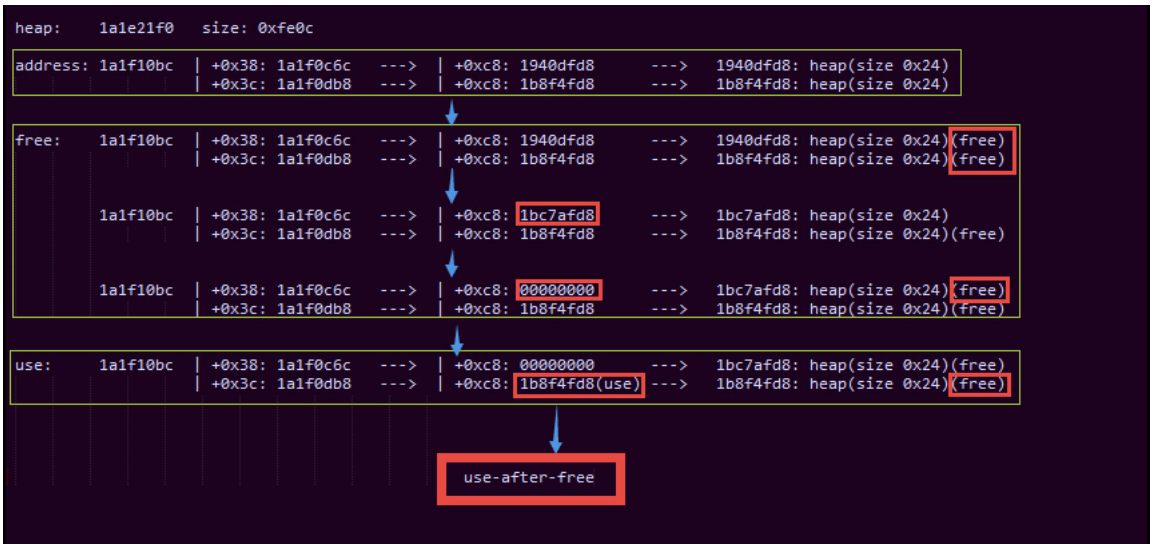

Figure 8. How the Heap Buffer is Created, Freed, then Reused

In short, because Adobe Reader DC fails to decompress the deflate-compressed data, it gets incomplete image data. Then, when it handles the data, a use-after-free issue is triggered.

Use-after-free vulnerabilities are often very complex and their causes vary from case to case. The common scenario works like this:

1. At some point an object is created with a vtable,
2. Later, the object gets called by a vtable pointer. If the object is freed before it gets called, the program will crash when it tries to call the object.

To exploit this kind of vulnerability, you normally need to perform the following steps:

1. At some point an object is created with a vtable,
2. Trigger a free operation on this object,
3. Create a fake object which resembles the freed object as closely as possible,
4. Later, when the vtable pointer is called, the fake object will be accessed and gain code execution.

Mitigation

All users of Adobe Acrobat and Reader are encouraged to upgrade to the latest version of this software. Additionally, organizations that have deployed Fortinet IPS solutions are already protected from this vulnerability with the signature Adobe.Acrobat.Reader.DC.Memory.Corruption.

by **Kai Lu and Kushal Arvind Shah** | Jun 06, 2016 | Filed in: Security Research (/category/security-research)

Tags: adobe (/tag/adobe)

⬆ Next Post: Turning Network Security Inside Out (/2016/06/06/turning-network-security-inside-out)

⬇ Previous Post: Fortinet Offers Carriers Practical Security Solutions for Today, with an Eye on Tomorrow (/2016/06/06/fortinet-offers-carriers-practical-security-solu