**CVE-2015-5122漏洞利用分析**

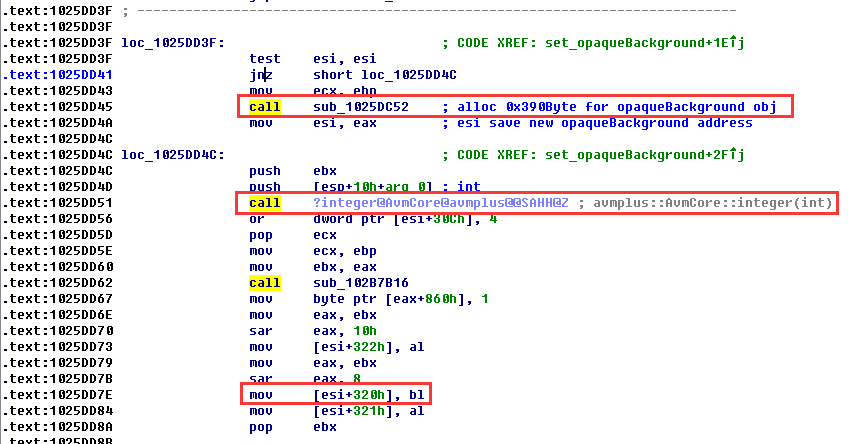
**0x00 调试环境**

|  |  |
| --- | --- |
| OS: | win7 x86 |
| IE: | ie10 |
| Flash: | flashplayer18\_0\_0\_203 |

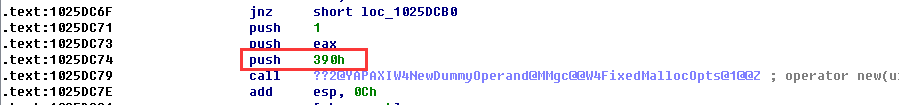
**0x01 漏洞成因**

CVE-2015-5122是HackingTeam数据泄露暴露出的另外一个0day，该漏洞是DisplayObject对象设置opaqueBackground属性时的一个UAF漏洞。

设置opaqueBackground属性的关键代码如下：



首先会判断DisplayObject对象是否存在opaqueBackground属性，如果不存在则调用sub\_1025DC52分配0x390 Byte的内存空间给opaqueBackground：



然后将new出的opaqueBackground地址保存至esi（在高级语言层面就是将指针保存给局部变量，很显然如果后面有机会释放这个指针指向的地址空间，那么这个指针就成为了一个悬挂指针）；

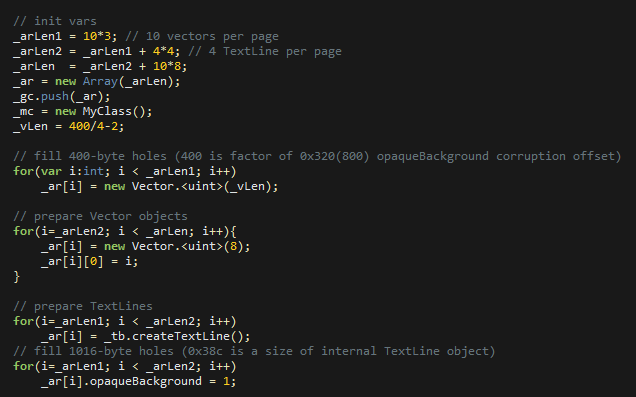
接着调用avmplus::AvmCore::integer(int) 函数为opaqueBackground属性赋值（如果赋值的是一个对象，则会调用这个对象的valueOf函数，显然valueOf函数可以自定义）；

最后通过mov [esi+320h], bl给这个指针偏移0x320的地址复制4Bytes。就如前面所说，如果能够将这个指针变成悬挂指针，我们就能修改这个指针偏移0x320处4Bytes的内存，从而造成内存破坏。

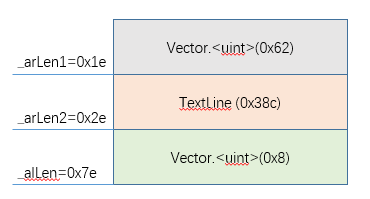
**0x02 漏洞利用**

**1) HeapSpray**

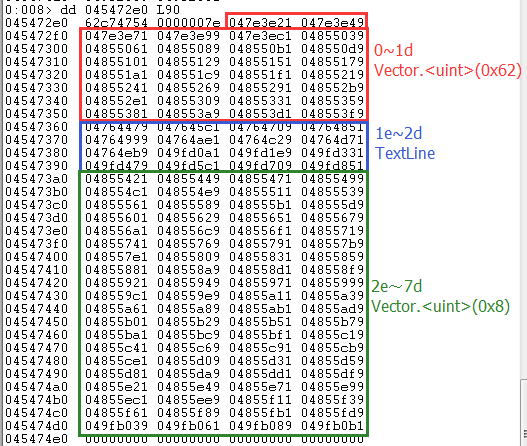
Exploit程序的HeapSpray代码如下：



\_ar由三部分组成：Vecor.<uint>(0x62)，TextLine对象，Vecor.<uint>(0x8)，具体排布如下：

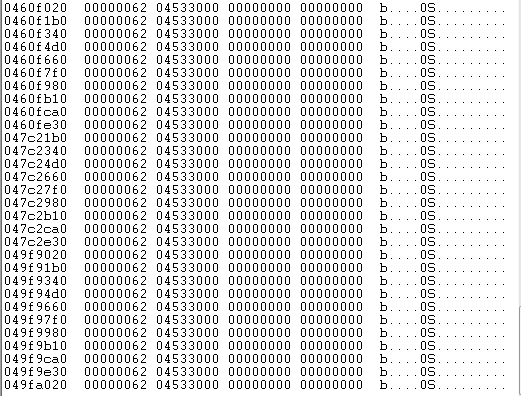


其中0～0x1d为Vecor.<uint>(0x62)，其data占8+0x62\*4 = 0x190Byte，0x1e~0x2d为TextLine对象，0x2e~0x7e为Vecor.<uint>(0x8)：

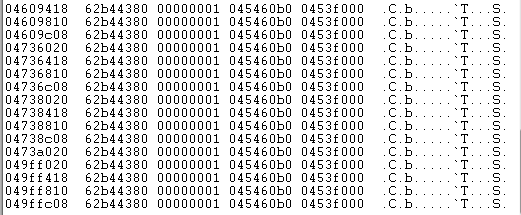


并通过给opaqueBackground对象赋值来分配内存。HeapSpray完成后，三部data内存分布：

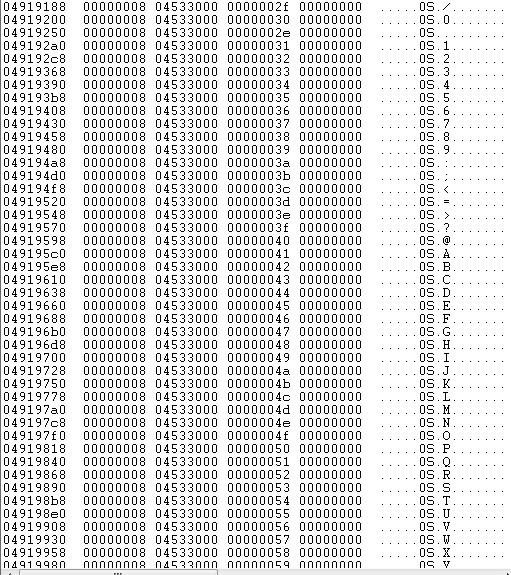
Vecor.<uint>(0x62) Data：



opaqueBackground Data：

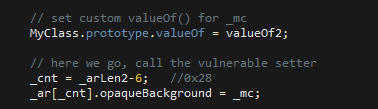


Vecor.<uint>(0x8) Data：

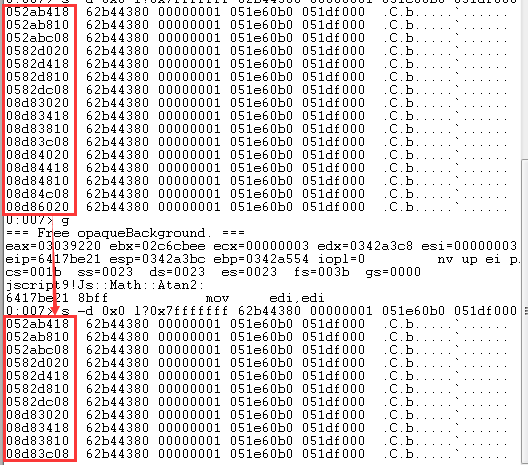


**2) 空洞、占位**

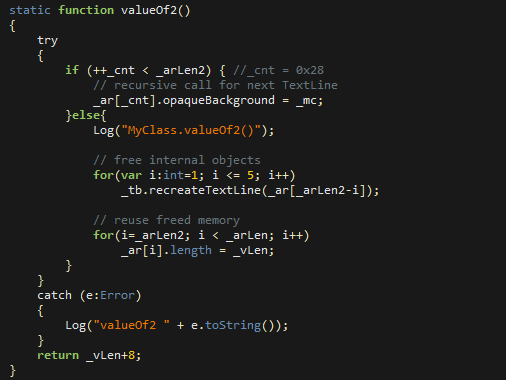
HeapSpray后，通过如下代码进入自定义valueOf函数，用来触发UAF漏洞：



接着在自定义valueOf函数，首先通过递归释放最后5个TextLine对象，这时候TextLine对象的opaqueBackground属性对象也会被释放：

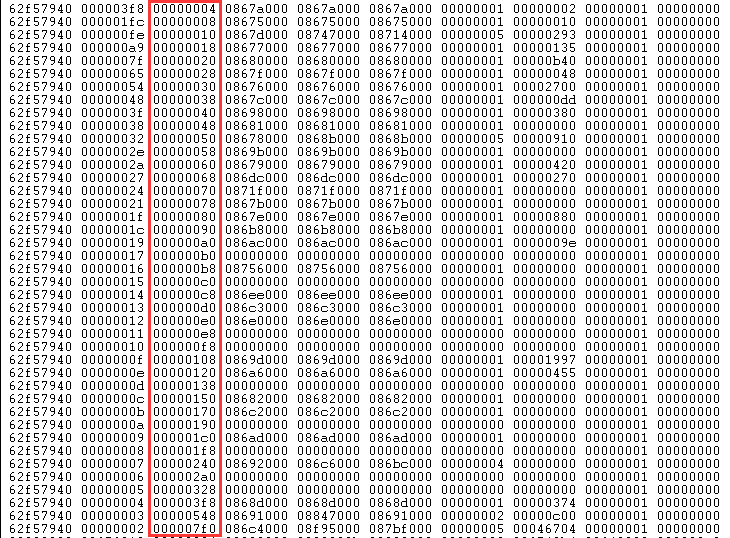


从而造成指向opaqueBackground对象释放内存的悬挂指针，接着对之前Vecor.<uint>(0x8)重新定义长度，占位opaqueBackground对象释放内存：

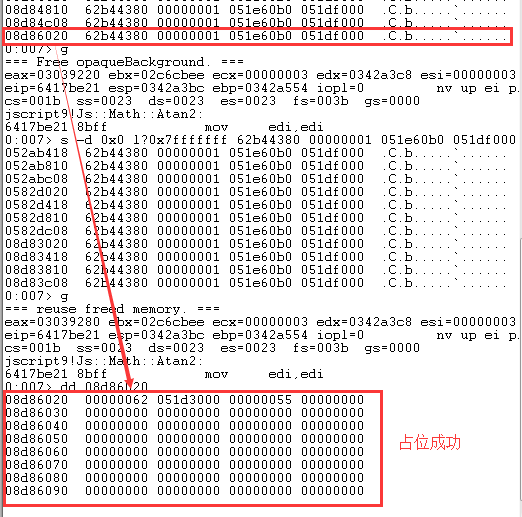


这里有个问题：opaqueBackground占位内存0x390Bytes，Vecor.<uint>(0x8)重新分配后Vecor.<uint>(0x62)占用内存0x190 Bytes，为什么Vecor.<uint>(0x62)会占位 0x390 Bytes的内存，而不是新申请的内存？

其实，根据flash有自己的custom heap管理，对于0x7F0大小以下的内存块，会由flash自己的custom heap管理（原理类似windows的前端堆），具体Block分配如下：

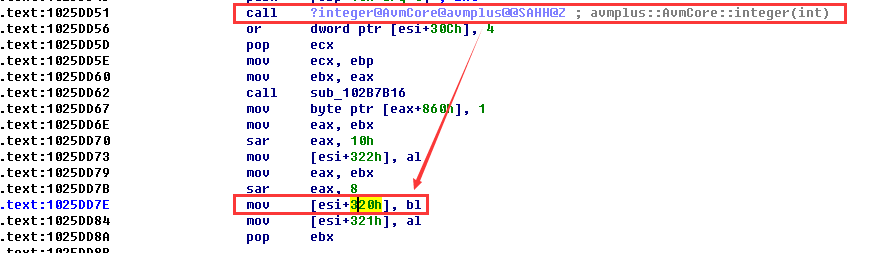


其中红框即为每个block的大小，所以opaqueBackground占位内存0x390Bytes会对齐到0x38f，Vecor.<uint>(0x62)会对齐到0x190，但是申请0x190依旧不会去占位0x38f的freelist，所以exploit程序为了实现占位，一次性释放了5个TextLine对象，同时5个opaqueBackground对象也会被释放，这里就释放了0x38f\*5=0x13D8>0x1000的内存，即一个完整的页内存被释放，所以Vecor.<uint>(0x62)会占位到这块内存。

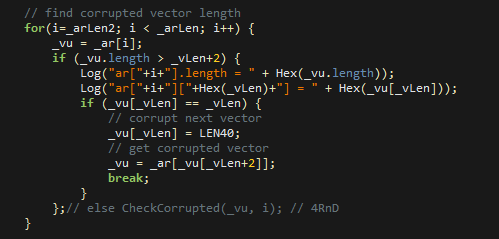


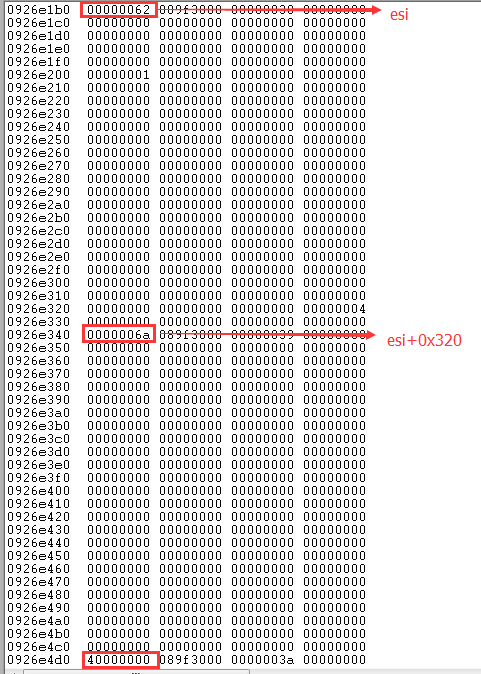
**3) 修改内存**

根据前面的分析，valueof的返回值会覆盖esi+0x320处的4Bytes值，0x320是Vecor.<uint>(0x62)的2倍，因此会修改某个Vecor.<uint>(0x62)的长度数据部分：



再通过这个被修改成长度0x6a的Vecor.<uint>去修改下个Vecor.<uint>(0x62)的长度部分，最终获得一个长度为0x4000000的Vecor.<uint>，从而获得了任意读写内存的权限：



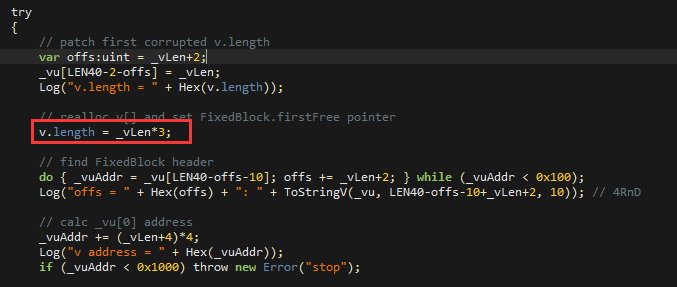


**4) 后续**

获得一个Vector.<uint>(0x40000000)后，后续利用主要需要做两件事：

（1）泄露该vector的VA，通过vector和索引构造任意地址读写函数

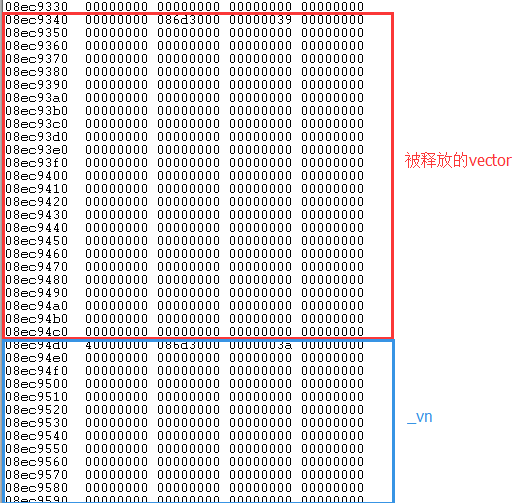
这里比较有趣的是泄露VA基址的方法，一般寻找\_uv的VA是通过保存相邻对象的this指针并通过读取this指针的值来泄露VA，这里的用法则是先通过重新修改相邻的vector的长度来释放该部分内存：



此时保存这块Block的BlockContainer的前4个Bytes会记录释放的内存地址，从而泄露了该块内存地址：



最终通过读取这个地址来获得\_vn的VA



（2）保存shellcode

（3）Bypass DEP，CFG，执行shellcode

CVE-2015-5122的后续利用可以参考CVE-2015-5119，因为是劫持了Jitcode生成的部分最后exploit效果

