**CVE-2015-0313漏洞利用分析**

**0x00 调试环境**

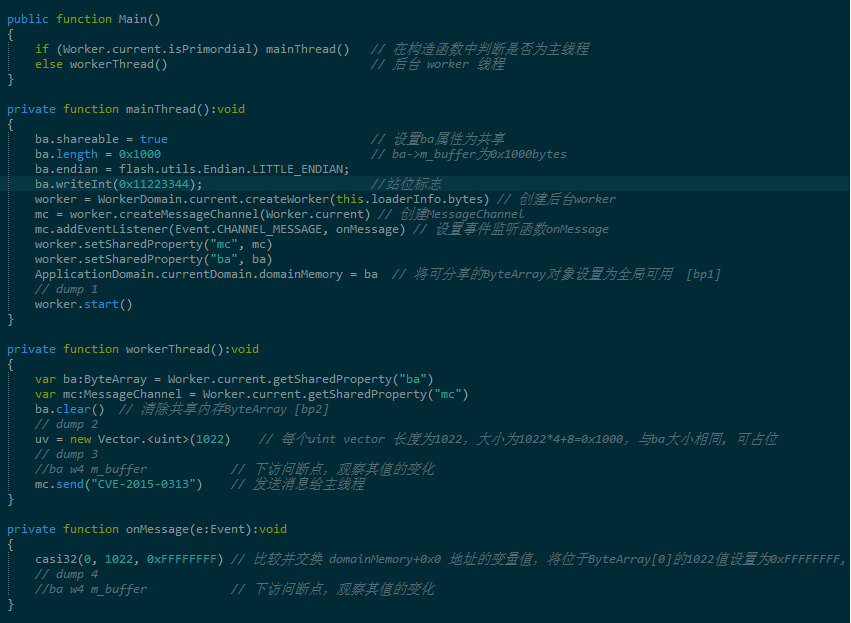
|  |  |
| --- | --- |
| OS: | win7 x86 |
| IE: | ie8 |
| Flash: | Flash32\_16\_0\_0\_296.ocx |

**0x01 漏洞成因**

**Worker**

AS3 Worker用于实现多线程操作，Worker彼此独立并且不访问相同的内存、变量和代码。Worker可以通过:MessageChannel、共享ByteArray（ba.shareable = true）和共享属性（worker.setSharedProperty("ba", ba)）三种方式实现数据传递。

**POC**

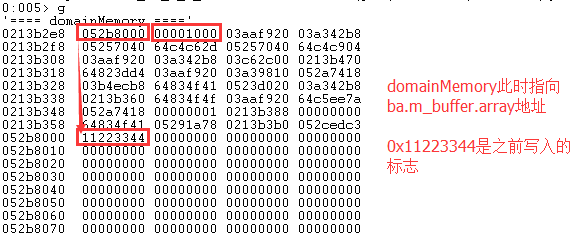


**流程**

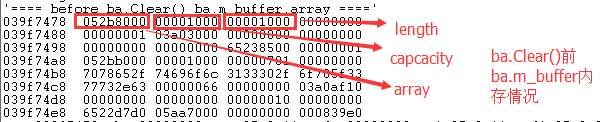
1. 主线程创建Worker线程
2. 主线程和Worker线程共享ba，并将domainMemory指向ba
3. Worker线程clear ba，并用一个Vector.<uint>占位，此时domainMemory为悬垂指针
4. domainMemory保持对原ba地址的引用，从而修改占位Vector.<uint>结构体信息（长度）

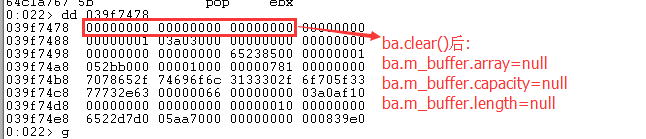
**调试**

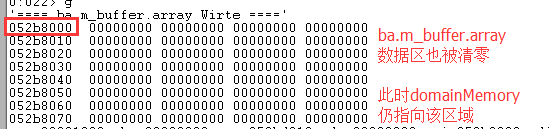
1. ApplicationDomain.currentDomain.domainMemory = ba后内存情况：



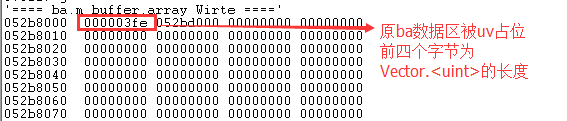
1. ba.clear()前后内存情况：



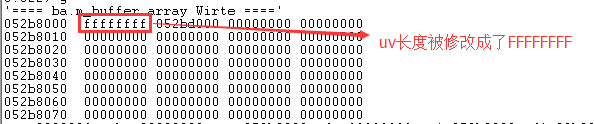




1. uv = new Vector.<uint>(1022) 占位(内存大小：4+4+0x3fe\*4 = 0x1000)



1. casi32(0, 1022, 0xFFFFFFFF)，通过domainMemory修改ba[0]值为0xFFFFFFFF，因为ba已被Worker线程clear,这里修改的是占位uv的前四个字节即Vector.<uint>的长度:



这里我们就获得了一个长度为0xFFFFFFFF的Vector.<uint>。

**0x02 漏洞利用**

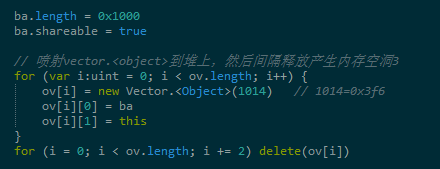
获得了一个长度为0xFFFFFFFF的uv后，需要计算该uv的va，再通过该uv实现任意地址内存读写。

**HeapSpray**

首先声明一个长度为25600(0x6400)的Vector.<Object>：



然后在主线程中heapspray:



Heapspray结束内存布局:

ov对象布局：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ov  header | ov[0] | ov[1] | ov[2] | ov[3] | … | ov[ox63FF] |

ov[i]对象布局：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ov[0]  header | ov[0][0]  ba | ov[0][1]  this | … | ov[0][3f5] | ov[1]  header | ov[1][0]  ba | ov[1][1]  this | … | ov[0x3F5]  header | … |

ov[i][0]的ba的m\_buffer指向一块大小为0x1000 byte内存，domainMemory也指向这块内存：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ba[0] | ba[1] | ba[2] | … | ba[0xfff] |
| domainMemory | | | | |

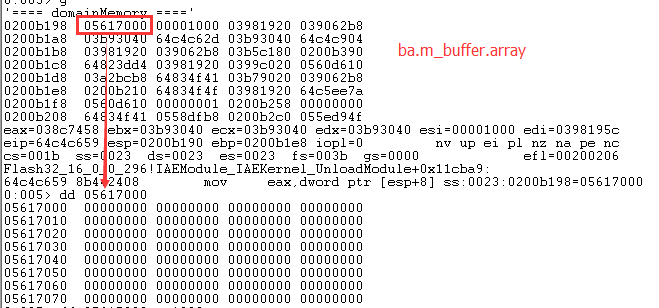
接着释放ov的偶数索引的元素（ov对象的灰色区域），形成空洞。

Worker线程清除ba，紧接着用一个大小为0x1000byte的uv占位ba清除后的空洞，同时将ov[0]指向该uv：

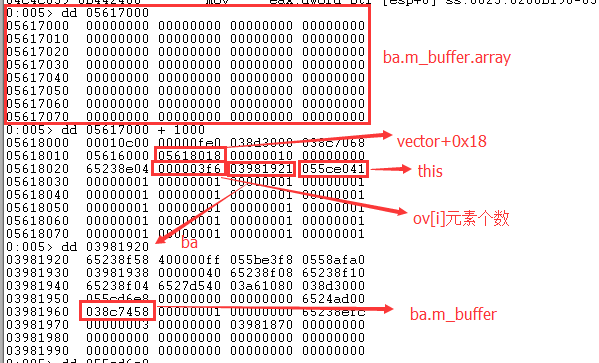


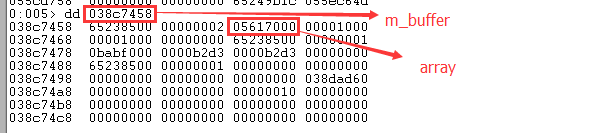
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ~~ba[0]~~ | ~~ba[1]~~ | ~~ba[2]~~ | ~~…~~ | ~~ba[0xfff]~~ |
| 0x3FE | xxxx | ov[0][0] | … | ov[0][3FD] |
| domainMemory | | | | |

调试情况（断在ApplicationDomain.currentDomain.domainMemory = ba）：



这里0x05617000为ba.m\_buffer.array地址，顺便看下后面0x1000是什么：





可以发现，ba后面紧跟着的是一个ov[i]，这样就可以通过被修改长度的ov[0]越界读取后面ov[i]的数据了。

**获得ov[0][0]的VA**

由heapspray部分分析可知，获得FFFFFFFF长度的ov[0]后面紧跟着某个ov[i]，对于Vector.<Object>其偏移0x14处为该vector地址+0x18，ov[0][0x403]即指向该地址，所以ov[0][0]的地址由如下方法获得：



**获得存放rop+shellcode的buffer VA**

由heapspray部分分析可知，ba地址在ov[i]偏移0x28处，即ov[0x408]：



注意到ov[0][0x408]获取的ba地址是avm的atom类型，其后3bit表示对象指针数据类型，因此其真实值为atom(addr)&0xFFFFFFF8。

**获得ov[i]对象的vtable**

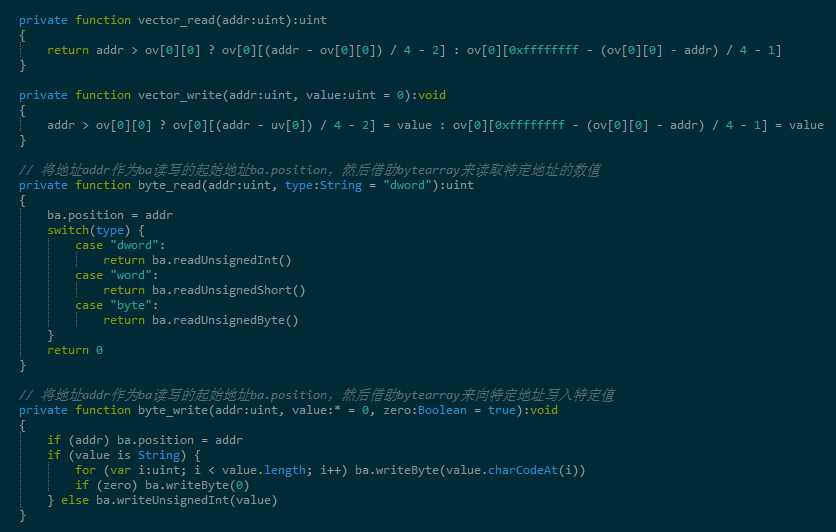
在heapspray中我们将ov[i][1] = this，因此其vtable内存地址通过下面方法获得：





函数vector\_read功能为根据传入地址读取其内存值

**内存读写函数实现**



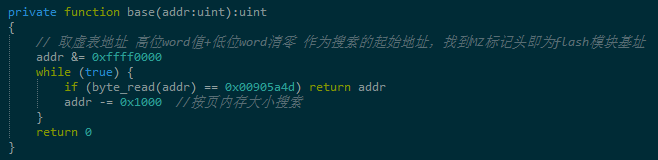
vector\_read()/ vector\_write()原理是通过将addr与ov[0][0]内存偏移转换为ov[0]的索引来实现内存的读写。

在实现了vector\_read()/ vector\_write()后修改了ov[0]下一个ov[i]的ba的长度为0xFFFFFFFF实现全地址内存读写：

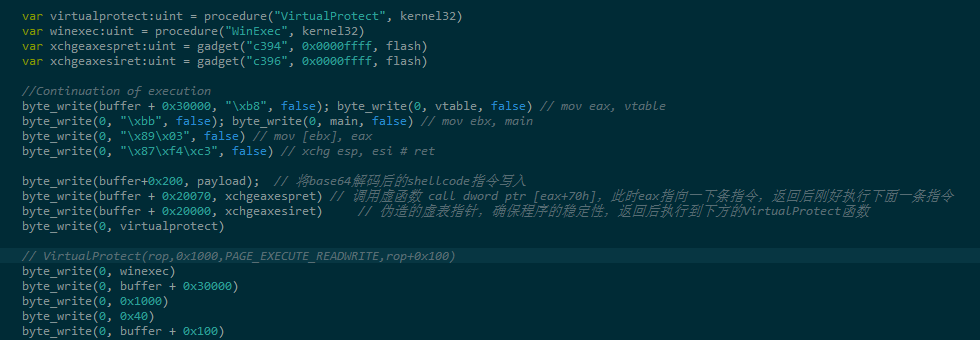


**Bypass ASLR**

前面已经获得了ov[i]对象的vtable，通过暴力搜索PE头即可获得flash基址：



**Bypass DEP**



**控制EIP**

