**CVE-2015-8651漏洞利用分析**

**0x00 调试环境**

|  |  |
| --- | --- |
| OS: | win7 x86 |
| IE: | ie10 |
| Flash: | flashplayer18\_0r0\_209\_win\_debug |

**0x01 背景**

之前在Freebuf看到一篇安天追影发表的文章：《暗黑客栈CVE-2015-8651漏洞原理分析》（http://www.freebuf.com/articles/network/93516.html），文中有对CVE-2015-8651漏洞原理的详细分析。苦于无法拿到文章中的样本，且个人水平有限，不能根据文章理解其漏洞原理。特别是漏洞触发关键代码描述略简：

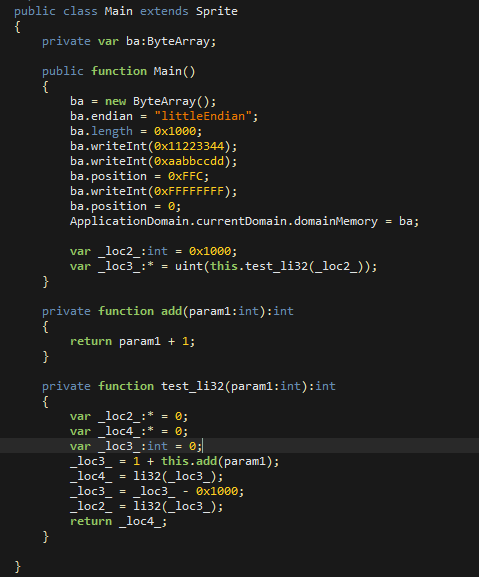


红框处的文字让我苦思冥想，构造多种POC，均无法复现漏洞现场。好在最近拿到的某EK中有该CVE的样本，方才了解其漏洞原理。

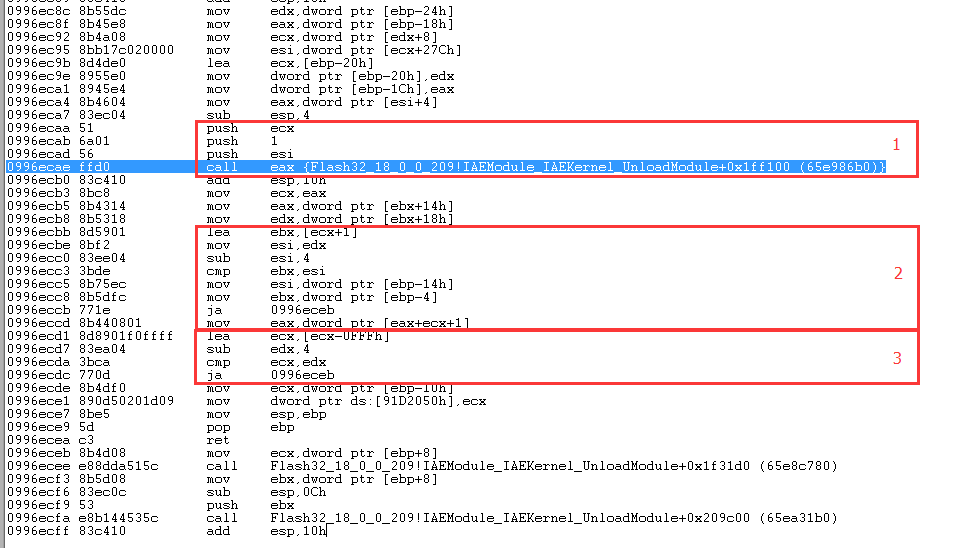
**0x02 漏洞原理**

DomainMemory以及li\*/si\*等方法的使用不再详述，可以参考安天的那篇文章。需要注意的是li\*/si\*等方法均由as3 code实现，而不是native code，运行后会由avm2生成jited code。

以li32函数为例，看下正常情况下生成的jited code的逻辑。poc代码：



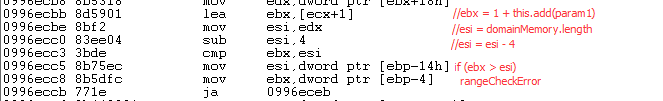
poc首先初始化ba，并赋值给domainMemory。关键部分在test\_li32方法中，调用了两次li32，传入的参数分别为\_loc3\_=1+this.add(param1)和\_loc3\_=\_loc3\_-0x1000，test\_li32方法jited后的代码如下：



说明：

1）1处对应poc中this.add(param1)的调用，因为是第一次调用add方法，call eax进入verifyjit流程，后面如果再次调用add方法则会直接调用add方法对应的jited code；

2）2处对应poc中\_loc3\_ = 1 + this.add(param1); \_loc4\_ = li32(\_loc3\_)的流程，li32方法会对输入参数进行长度校验，因为是从当前地址开始往后读4Bytes，需要对domainMemory的长度-4后再进行比较，详细分析如下：



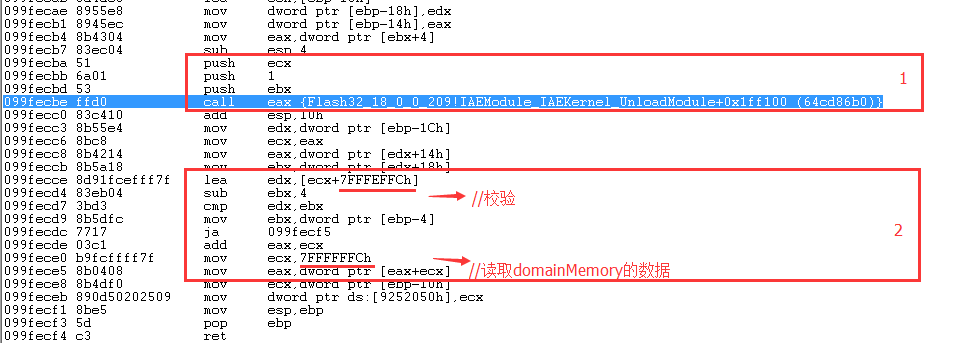
3）3处对应poc中：\_loc3\_ = \_loc3\_ - 0x1000; \_loc2\_ = li32(\_loc3\_);其中li32返回值赋给\_loc2\_，因为只是局部变量后续并未使用，jited code优化后没有2）校验后的赋值操作，其余与2）同理。

综上分析，li32会对输入进行长度校验，具体做法就是取domainMemory长度-4，和输入进行比较，如果输入的数据> domainMemory长度-4，则抛出异常，不再读取domainMemory里的数据。

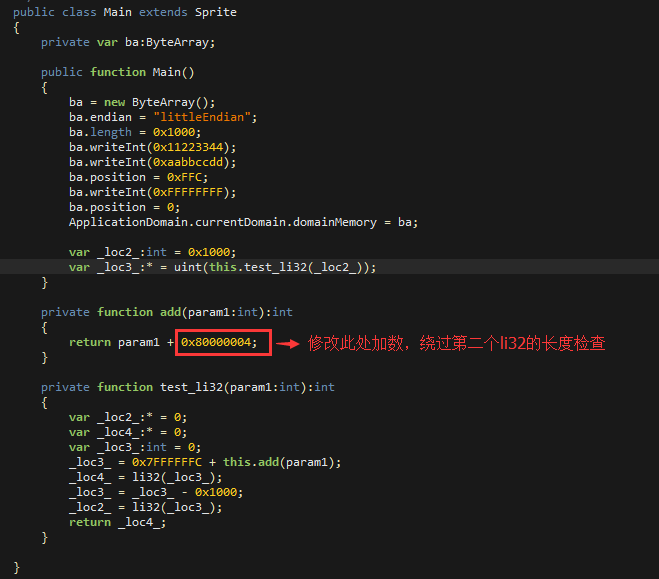
现在对poc代码稍作修改，修改点只有一处，将\_loc3\_ = 1 + this.add(param1);修改为\_loc3\_ = 0x7FFFFFFC + this.add(param1);



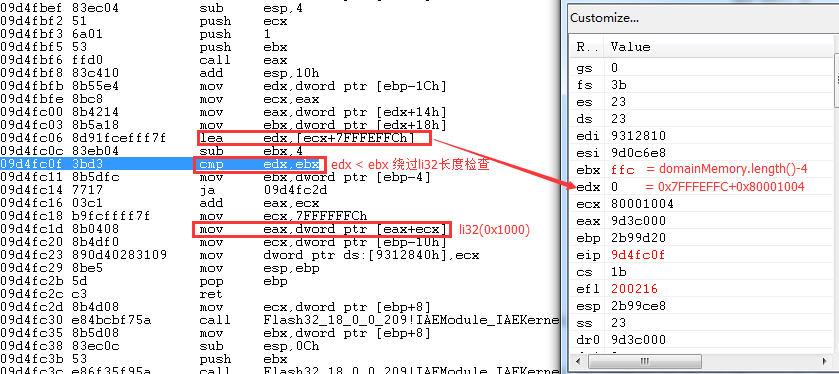
再来观察生成的Jited code：



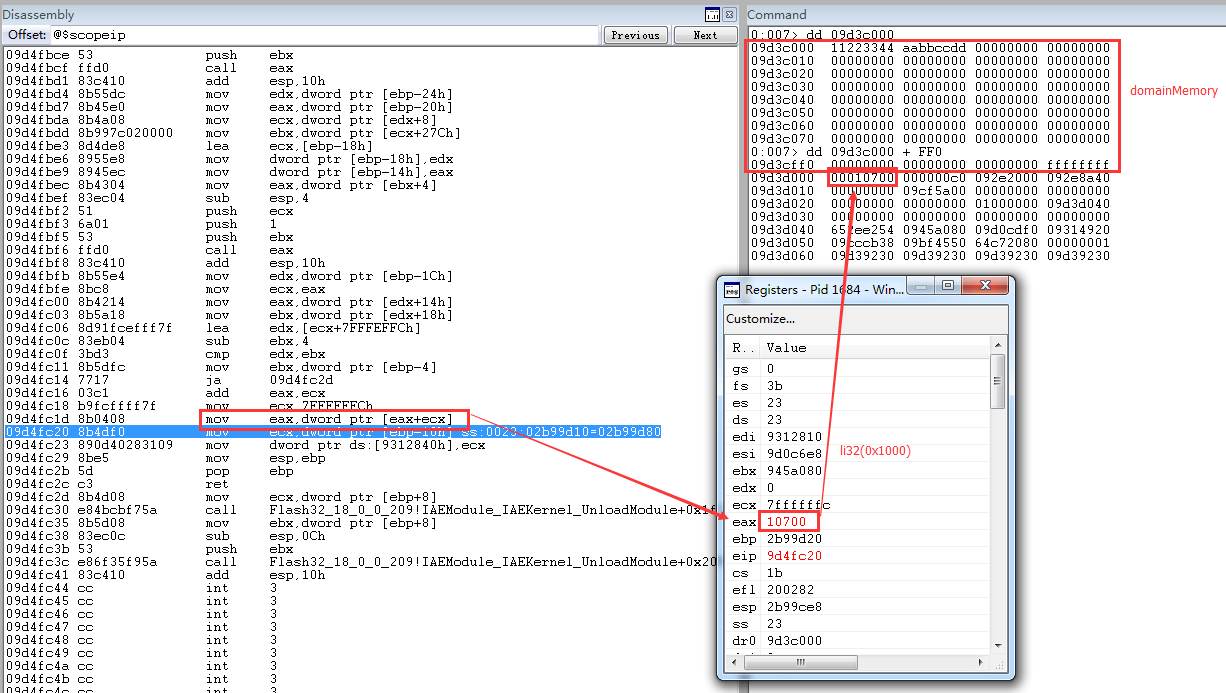
这里发现修改过的poc在jit后li32的校验被优化成了一个，并且校验的是第二个li32方法！也就是说我们可以利用这个漏洞可以通过第一个li32越界读取domainMemory以外的数据，只要保证越界的地址-0x1000能够通过第二个li32的校验。于是就有了最终的poc：



这里把add方法里的加数修改为0x80000004，其目的是add方法的返回值0x80001004+0x7FFFFEFC=0，从而绕过第二个li32的长度检查，调试过程：



最终利用漏洞成功读取domainMemory外的内存数据：



同理si32也存在相同漏洞，利用漏洞可以修改domainMemory外的内存数据。后面的利用方法一般都是利用HeapSpray排布Vector或者ByteArray对象，通过修改长度属性实现任意地址读写，不再说明。