VUPEN security

VUPEN Security – Binary Analysis & Exploits Service

In-Depth Analysis of Microsoft Office Word Record Parsing Buffer Overflow Vulnerability (MS10-056 / CVE-2010-1900)

Table of Contents

Introduction

A vulnerability exists in Microsoft Office Word when processing Word97 documents, which could be exploited by attackers to execute arbitrary code via a malicious DOC file.

Tested Versions

The vulnerability was analysed on Windows XP SP3 with a Microsoft Office Word 2003 SP3 (winword.exe version 11.0.8324.0).

Fixed Versions

The vulnerability was fixed with the MS10-056 security update.

Technical Details

Microsoft Word allocates a large stack buffer and copies the content of the various property modifiers there. However, the function responsible for parsing "sprmCMajority" (opcode 0xCA47) does not check if the buffer is large enough to hold the data contained within this SPRM (Single Property Modifier). Malicious documents can thus trigger an exploitable stack overflow which can allow arbitrary code to be executed by an attacker.

First of all, a large stack buffer is declared in sub_3028B449:

```
.text:3028B449              push    ebp
.text:3028B44A              mov     ebp, esp
.text:3028B44C              mov     eax, 40ACh
.text:3028B451              call    sub_30006781
.text:3028B456              mov     eax, ds:dword_30B84000
.text:3028B45B              push    3FA3h                       //max_size
.text:3028B460              push    [ebp+arg_24]
.text:3028B463              mov     [ebp+var_4], eax
.text:3028B466              push    [ebp+arg_20]
.text:3028B469              lea     eax, [ebp+var_40AC]         //stack buffer
.text:3028B46F              push    [ebp+arg_1C]
.text:3028B472              push    0
.text:3028B474              push    [ebp+arg_18]
.text:3028B477              push    [ebp+arg_14]
.text:3028B47A              push    eax
.text:3028B47B              lea     eax, [ebp+arg_4]
.text:3028B47E              push    eax
.text:3028B47F              push    [ebp+Src]
.text:3028B482              call    sub_3028AFEC                //parse various sprms
```

The stack buffer declared here is filled with the sprms and parameters encountered by the parser. Each time a sprm is parsed, its content is copied to this buffer.

The "sprmCMajority" property modifier is specifically parsed in sub_3093A168:

```
.text:304DA220 loc_304DA220:
.text:304DA220              cmp     [ebp+var_554], 0CA47h       //case sprmCMajority
.text:304DA22A              jz      loc_304DAC73
…
.text:304DAC73 loc_304DAC73:
.text:304DAC73              cmp     [ebp+arg_14], 0
.text:304DAC77              jnz     short loc_304DAC9A
.text:304DAC79              push    [ebp+arg_20]
.text:304DAC7C              push    [ebp+arg_1C]
.text:304DAC7F              push    [ebp+arg_18]
```

```
.text:304DAC82          push    [ebp+arg_10]
.text:304DAC85          push    [ebp+arg_C]
.text:304DAC88          lea     eax, [ebp+Dst]
.text:304DAC8B          push    eax
.text:304DAC8C          lea     eax, [ebp+Src]
.text:304DAC8F          push    eax
.text:304DAC90          call    sub_3093A168            //parse sprmCMajority
```

The vulnerability occurs in this function:

```
.text:3093A168          push    ebp
.text:3093A169          mov     ebp, esp
.text:3093A16B          sub     esp, 698h
.text:3093A171          mov     eax, ds:dword_30B84000
.text:3093A176          mov     [ebp+var_4], eax
.text:3093A179          push    edi
.text:3093A17A          mov     eax, [ebp+arg_0]
.text:3093A17D          mov     eax, [eax]              //eax points to the length of the
                                                        //parameter of the sprmCMajority
.text:3093A17F          movzx   eax, byte ptr [eax]
.text:3093A182          mov     [ebp+var_494], eax      //save this length to var_494
.text:3093A188          cmp     [ebp+arg_8], 4Eh        //compare nFib with 4Eh
.text:3093A18C          jl      short loc_3093A200
.text:3093A18E          push    485h
.text:3093A193          push    [ebp+var_494]
.text:3093A199          call    sub_30045AE9            //return min(485h, length) = length
                                                        //because length < 100h
.text:3093A19E          mov     [ebp+var_494], eax      //move length to var_494
.text:3093A1A4          push    [ebp+var_494]
.text:3093A1AA          lea     eax, [ebp+var_490]
.text:3093A1B0          push    eax
.text:3093A1B1          mov     eax, [ebp+arg_0]
.text:3093A1B4          mov     eax, [eax]
.text:3093A1B6          inc     eax
.text:3093A1B7          push    eax
.text:3093A1B8          call    memcpy                  //copy the content of this sprm to var_490
.text:3093A1BD          mov     ax, word ptr [ebp+var_494]
.text:3093A1C4          mov     word ptr [ebp+var_498], ax
.text:3093A1CB          push    383h                    //max_size
.text:3093A1D0          push    [ebp+arg_18]
.text:3093A1D3          push    [ebp+arg_14]
.text:3093A1D6          push    [ebp+arg_10]
.text:3093A1D9          push    0
.text:3093A1DB          push    [ebp+arg_C]
.text:3093A1DE          push    [ebp+arg_8]
.text:3093A1E1          lea     eax, [ebp+var_490]
.text:3093A1E7          push    eax
.text:3093A1E8          lea     eax, [ebp+var_498]
.text:3093A1EE          push    eax
.text:3093A1EF          mov     eax, [ebp+arg_0]
.text:3093A1F2          mov     eax, [eax]
.text:3093A1F4          inc     eax
.text:3093A1F5          push    eax
.text:3093A1F6          call    sub_3028AFEC            //process the sprms found in
                                                        //sprmCMajority
.text:3093A1FB          jmp     loc_3093A2FF
```

The problem first lies at this point. Actually the data embedded in "sprmCMajority" consists of various sprms. As we can see, the program passes var_490 to sub_3028AFEC. This variable is actually filled with data read from the current "sprmCMajority".

For example the following bytes represent an "sprmCMajority" embedding an "sprmTDefTable" (0xD608).

```
47 CA FF 08 D6 F0 02
```

The vulnerable sprm is followed by FFh bytes. "sprmTDefTable" was specifically chosen in this example because it may be followed by a parameter larger than FFh bytes. Given that Word allows up to 0383h bytes to be copied to var_490, it is thus possible to insert 02F0h bytes in this "sprmTDefTable". When the function returns, var_498 is updated with the actual amount of data written.

The vulnerability actually occurs when the "sprmCMajority" follows enough sprms so that the stack buffer allocated in sub_3028B449 is almost filled when the vulnerable function is entered. Given that this buffer is 3FA3h bytes long, an issue may occur if 3FA3h – 383h = 3C20h bytes are already stored in the buffer. This issue stems then from the following lines:

```
.text:3093A33C loc_3093A33C:
.text:3093A33C              mov     eax, [ebp+arg_4]
.text:3093A33F              mov     eax, [eax]
.text:3093A341              mov     cl, byte ptr [ebp+var_498]
.text:3093A347              mov     [eax], cl
.text:3093A349              movsx   eax, word ptr [ebp+var_498]      //get the actual amount of data
.text:3093A350              push    eax
.text:3093A351              mov     eax, [ebp+arg_4]
.text:3093A354              mov     eax, [eax]
.text:3093A356              inc     eax
.text:3093A357              push    eax                              //destination buffer, declared in
                                                                      sub_3028B449
.text:3093A358              lea     eax, [ebp+var_490]
.text:3093A35E              push    eax
.text:3093A35F              call    memmove
```

At this point, if enough sprms have been provided before encountering a vulnerable sprmCMajority, an exploitable stack overflow occurs.

Exploitation

Exploitation of this vulnerability is trivial with Excel 2002. Given that this program is not compiled with the security features included in Excel 2003 and above, it is trivial to overwrite a return address of a calling function and thus execute an egghunter to find the payload located somewhere in memory.

With Excel 2003, the provided exploit takes advantage of various arguments overwritten in sub_3028B449:

```
.text:3028B47B              lea     eax, [ebp+arg_4]
.text:3028B47E              push    eax
.text:3028B47F              push    [ebp+Src]
.text:3028B482              call    sub_3028AFEC                 //the stack overflow is triggered here
.text:3028B487              movsx   edx, word ptr [ebp+arg_4]
.text:3028B48B              push    0
.text:3028B48D              push    [ebp+arg_38]                 //each of these arguments can be
                                                                   controlled
.text:3028B490              lea     ecx, [ebp+var_40AC]
.text:3028B496              push    [ebp+arg_34]
.text:3028B499              push    [ebp+arg_30]
.text:3028B49C              push    [ebp+arg_2C]
.text:3028B49F              push    [ebp+arg_28]
```

```
.text:3028B4A2              push    [ebp+arg_10]
.text:3028B4A5              push    [ebp+arg_C]
.text:3028B4A8              push    [ebp+arg_8]
.text:3028B4AB              call    sub_30038555
```

This function performs various actions on the sprms encountered. Sprms 6412h to 641Ah are very interesting because they allow multiple four bytes overwrite to an arbitrary location pointed by arg_0:

```
.text:30038639              push    4                       //counter
.text:3003863B
.text:3003863B loc_3003863B:
.text:3003863B              mov     ecx, [ebp+var_4]
.text:3003863E              mov     edx, esi
.text:30038640              shr     edx, 1
.text:30038642              and     edx, 0FFFh
.text:30038648              add     edx, [ebp+arg_0]        //arg_0 is controlled
.text:3003864B              add     ecx, 2                  //ecx points to controlled data
.text:3003864E              call    memmove
```

This exploit thus consists in overwriting the Safe Exception Handler located at 0x0012FFB0 with an arbitrary address. Once done, an exception can be triggered thanks to sprm 0x6654. This sprm is parsed in sub_3004786C:

```
.text:303A78DD              cmp     [ebp+var_20E4], 6654h   //case 6654h
.text:303A78E7              jz      loc_3036C524
…
.text:3036C524 loc_3036C524:
.text:3036C524              push    4                       //amount of bytes to write
.text:3036C526              mov     eax, [ebp+var_54]       //var_54h points to 0x0012FFA0
.text:3036C529              add     eax, 0CCh
.text:3036C52E              push    eax
.text:3036C52F              push    [ebp+var_58]
.text:3036C532              call    memcpy
```

An exception is then triggered because 0x0012FFA0 + CCh points outside of the scope of the current stack. The flow is then redirected to "pop pop retn" in MSCOMCTL.OCX and the next code is executed:

```
b8 f9 f8 f7 f6      mov eax, 0xF6F7F8F9
start:
59                  pop ecx
39 c1               cmp ecx, eax
75 FB               jne start
FF E4               jmp esp
```

At this point the program executes an egghunter and eventually the payload. This exploit has been tested with MSCOMCTL.OCX version 6.1.95.45 provided by default with Microsoft Office XP and Office 2003.

Exploitation is far more difficult with Word 2007 due to a stack cookie which seems difficult to bypass.

Detection

It is difficult to reliably detect an exploit for this vulnerability as the "sprmCMajority" is vulnerable only if it is preceded by a large amount of sprms.

The best way to detect an exploit taking advantage of this vulnerability is to analyse each grpprl (group of sprms) embedded in a Word document. Then, check if they carry a sprmCMajority (sprm 0xCA47) and note its offset.

To avoid a consequent amount of false positives, consider that a file carrying a "sprmTDefTableShd" at an offset greater than 0x2000 in a grpprl is suspicious. Word documents usually do not contain so many sprms in the same grpprl so 0x2000 should be a good threshold to detect a malicious file.

References

VUPEN/ADV-2010-2053:
http://www.vupen.com/english/advisories/2010/2053

MS10-056:
http://www.microsoft.com/technet/security/bulletin/ms10-056.mspx

Changelog

2010-08-24: Initial release