



# SA30285 / CVE-2008-4024

**Generated by Secunia**

**17 December, 2008**

*6 pages*

## Table of Contents

<b>Introduction</b>	<b>2</b>
<b>Technical Details</b>	<b>2</b>
<b>Exploitation</b>	<b>5</b>
<b>Characteristics</b>	<b>6</b>
<b>Tested Versions</b>	<b>6</b>
<b>Fixed Versions</b>	<b>6</b>
<b>References</b>	<b>6</b>

## Introduction:

=====

A vulnerability in Microsoft Word can be exploited by malicious people to compromise a user's system when processing list formatting structures in DOC files.

## Technical Details:

=====

Insufficient input validation when parsing list formatting structures in DOC files can result in an attacker-controlled pointer being passed to a heap free() function.

At the beginning of the document stream in a Microsoft Word document is the documented File Information Block (FIB), which contains information about the remainder of the file. A pair of values indicate the location (fcPlfLfo) and size (lcbPlfLfo) of some "list format information".

The list format information consists of an array of 16-byte LFO structures in the file, which are defined as:

```
List Format Override (LFO)
b10 b16 Field      Type      Size  Comments
0   0x0 lsid       long      4     List ID of corresponding LSTF (see LSTF)
4   0x4 reserved   long      4     Reserved
8   0x8 reserved   long      4     Reserved
12  0xC clfolvl    uns char  1     Count of levels whose format is overridden (see LFOLVL)
13  0xD ibstFltAutoNum uns char  1     Used for AUTONUM field emulation
14  0xE grfhic     uns char  1     HTML compatibility flags
15  0xF reserved   uns char  1     Reserved
```

When parsing the FIB in sub\_30022AB2(), if lcbPlfLfo is non-zero then the offset to the data (fcPlfLfo) is passed to sub\_3024A628() along with a number of other arguments.

```
.text:30023413 loc_30023413:                ; CODE XREF: sub_30022AB2+22774B
.text:30023413                cmp     [ebp+lcbPlfLfo], edi
.text:30023419                jnz    loc_3024A5F4
...
.text:3024A5F4 loc_3024A5F4:                ; CODE XREF: sub_30022AB2+967
.text:3024A5F4                push   edi
.text:3024A5F5                push   edi
.text:3024A5F6                push   222h
.text:3024A5FB                push   10h
.text:3024A5FD                push   [ebp+fcPlfLfo]
.text:3024A603                push   2
.text:3024A605                push   [ebp+arg_0]
.text:3024A608                call   sub_3024A628
```

In sub\_3024A628(), the LFO data is read in by sub\_3002A523().

```

.text:3024A628 sub_3024A628 proc near          ; CODE XREF: sub_30022AB2+227B56
.text:3024A628                                ; sub_305569D7+ED9
.text:3024A628
.text:3024A628 tempLFO      = LFO ptr -14h
.text:3024A628 nFib        = dword ptr -4
.text:3024A628 arg_0       = dword ptr 8
.text:3024A628 arg_4       = dword ptr 0Ch
.text:3024A628 fcPlfLfo    = dword ptr 10h
.text:3024A628 arg_C       = dword ptr 14h
.text:3024A628 arg_10      = dword ptr 18h
.text:3024A628 arg_14      = dword ptr 1Ch
.text:3024A628 arg_18      = dword ptr 20h
.text:3024A628
.text:3024A628                push    ebp
.text:3024A629                mov     ebp, esp
.text:3024A62B                sub     esp, 14h
.text:3024A62E                push    ebx
.text:3024A62F                push    esi
.text:3024A630                mov     eax, g_pSomething
.text:3024A635                push    edi
.text:3024A636                xor     edi, edi
.text:3024A638                mov     ecx, [eax]
.text:3024A63A                mov     eax, [ebp+arg_0]
.text:3024A63D                push    edi
.text:3024A63E                push    [ebp+arg_10]
.text:3024A641                mov     ecx, [ecx+eax*4]
.text:3024A644                push    edi
.text:3024A645                push    10h
.text:3024A647                push    [ebp+arg_C]
.text:3024A64A                movsx   ecx, word ptr [ecx+80h]
.text:3024A651                push    [ebp+fcPlfLfo]
.text:3024A654                mov     [ebp+nFib], ecx
.text:3024A657                push    [ebp+arg_4]
.text:3024A65A                push    eax
.text:3024A65B                call   sub_3002A523    ; read in array of LFOs
.text:3024A660                mov     esi, eax
.text:3024A662                cmp     esi, edi
.text:3024A664                jz     @return_0      ; no array returned?

```

After some insignificant processing in sub\_3002A523(), the 32-bit value at the given offset is read and interpreted as the number of following LFO entries.

```

.text:3002A56D                lea    eax, [ebp+arg_4]
.text:3002A570                push   4
.text:3002A572                push   eax
.text:3002A573                push   ebx
.text:3002A574                push   esi
.text:3002A575                call   sub_300200BD    ; get size of array

```

An array is allocated with enough space for the given number of 16-byte entries (not shown in disassembly) and the data is directly read from the file into it.

```

.text:3002A62A                mov     eax, [ebp+arg_4] ; number of entries
.text:3002A62D                push   0
.text:3002A62F                imul  eax, [ebp+arg_C] ; multiply by size of each entry (16)
.text:3002A633                push   eax
.text:3002A634                push   edi
.text:3002A635                push   ebx
.text:3002A636                push   [ebp+arg_0]
.text:3002A639                call   sub_300200BD    ; read array data directly into array

```

A pointer to the array is returned to sub\_3024A628() (not shown in disassembly).

If the returned array is not empty, each entry is processed in a loop where the 16-byte entry is first copied onto the stack.

```

.text:3024A66A      mov     eax, [esi]
.text:3024A66C      mov     eax, [eax]
.text:3024A66E      cmp     eax, edi
.text:3024A670      mov     [ebp+arg_C], eax
.text:3024A673      jle     short @return_esi ; empty array?
.text:3024A675
.text:3024A675 @loop_process_LFOs:                ; CODE XREF: sub_3024A628+CA
.text:3024A675      mov     eax, [esi]
.text:3024A677      mov     edx, edi          ; index
.text:3024A679      mov     ecx, esi
.text:3024A67B      mov     ebx, [eax+8]
.text:3024A67E      call   sub_30004E59      ; get pointer to array entry
.text:3024A683      push   ebx
.text:3024A684      lea   edx, [ebp+tempLFO]
.text:3024A687      mov     ecx, eax
.text:3024A689      call   fastcall_memcpy ; copy LFO entry onto stack

```

After some insignificant processing in the loop (not shown in disassembly), a pointer to the copy on the stack is passed to sub\_3024A6FD().

```

.text:3024A6B8      push   [ebp+arg_18]
.text:3024A6BB      lea   eax, [ebp+tempLFO]
.text:3024A6BE      push   [ebp+arg_14]
.text:3024A6C1      push   [ebp+arg_4]
.text:3024A6C4      push   [ebp+arg_0]
.text:3024A6C7      push   eax
.text:3024A6C8      call   sub_3024A6FD

```

In sub\_3024A6FD(), if the clfolvl field in the LFO is zero then the function simply returns.

```

.text:3024A6FD sub_3024A6FD  proc near                ; CODE XREF: sub_3024A628+A0
.text:3024A6FD
.text:3024A6FD arg_0      = dword ptr 8
.text:3024A6FD arg_4      = dword ptr 0Ch
.text:3024A6FD arg_8      = dword ptr 10h
.text:3024A6FD arg_C      = dword ptr 14h
.text:3024A6FD arg_10     = dword ptr 18h
.text:3024A6FD
.text:3024A6FD      push   ebp
.text:3024A6FE      mov     ebp, esp
.text:3024A700      push   ebx
.text:3024A701      mov     ebx, [ebp+arg_0]
.text:3024A704      push   esi
.text:3024A705      push   edi
.text:3024A706      mov     al, [ebx+LFO.clfolvl]
.text:3024A709      test   al, al
.text:3024A70B      jz     short @return_1

```

In the case where clfolvl is non-zero, memory is allocated on the heap and a pointer to it stored in the second reserved field of the LFO at offset eight.

```

.text:3024A70D      mov     edi, _MsoPvAllocCore@8 ; MsoPvAllocCore(x,x)
.text:3024A713      push   2
.text:3024A715      movzx  eax, al
.text:3024A718      shl     eax, 3
.text:3024A71B      push   eax
.text:3024A71C      call   edi ; MsoPvAllocCore(x,x) ; MsoPvAllocCore(x,x)
.text:3024A71E      mov     esi, eax
.text:3024A720      test   esi, esi
.text:3024A722      jz     @return_0          ; error allocating?
.text:3024A728      movzx  eax, [ebx+LFO.clfolvl]
.text:3024A72C      mov     [ebx+LFO.reserved2], esi

```

Upon returning to sub\_3024A628(), the (possibly modified) LFO on the stack is copied back into the array.

```
.text:3024A6D5      mov     eax, [esi]
.text:3024A6D7      mov     edx, edi
.text:3024A6D9      mov     ecx, esi
.text:3024A6DB      mov     ebx, [eax+8]
.text:3024A6DE      call    sub_30004E59    ; get pointer to array entry
.text:3024A6E3      push   ebx
.text:3024A6E4      mov     edx, eax
.text:3024A6E6      lea    ecx, [ebp+tempLFO]
.text:3024A6E9      call    fastcall_memcpy ; copy modified stack copy back into array
```

At a later time during processing or when closing the file, the array of LFOs is destructed and freed.

For each array entry being freed, sub\_3022B9B7() is called with a pointer to the LFO structure

```
.text:3022B9B7 sub_3022B9B7  proc near          ; CODE XREF: sub_30140308+EB616
.text:3022B9B7
.text:3022B9B7 arg_0      = dword ptr  4
.text:3022B9B7
.text:3022B9B7      push   ebx
.text:3022B9B8      push   ebp
.text:3022B9B9      push   esi
.text:3022B9BA      push   edi
.text:3022B9BB      mov     edi, [esp+10h+arg_0]
.text:3022B9BF      mov     ebp, _MsoFreePv@4 ; MsoFreePv(x)
.text:3022B9C5      movzx  eax, [edi+LFO.clfolvl]
.text:3022B9C9      mov     esi, [edi+LFO.reserved2]
.text:3022B9CC      lea    ebx, [esi+eax*8]
.text:3022B9CF      cmp     esi, ebx
.text:3022B9D1      jb     short loc_3022B9EF ; clfolvl non-zero?
```

Eventually, the reserved2 field is checked against zero and, if non-zero, it is passed as the pointer to MsoFreePv().

```
.text:3022B9D3
.text:3022B9D3 loc_3022B9D3:          ; CODE XREF: sub_3022B9B7+49
.text:3022B9D3      mov     eax, [edi+LFO.reserved2]
.text:3022B9D6      test    eax, eax
.text:3022B9D8      jnz    short loc_3022BA02 ; reserved2 non-zero?
...
.text:3022BA02 loc_3022BA02:          ; CODE XREF: sub_3022B9B7+21
.text:3022BA02      push   eax
.text:3022BA03      call   ebp ; MsoFreePv(x) ; MsoFreePv(x)
```

If clfolvl for a given entry was zero in the file, then the reserved2 value passed to MsoFreePv() is the value taken directly from the file with no validation.

#### Exploitation:

=====

By convincing a user to open a Word document that contains a malicious LFO entry (clfolvl set to zero), an attacker can control the pointer passed to MsoFreePv(). Arbitrary code execution is thus considered possible by leveraging the controlled free() to cause a four byte write.

Secunia has developed a PoC for the vulnerability, which is available via the BA customer web interface.

## Characteristics:

=====

## Detection:

Look for a binary Word (97-2007) file where lcbPlfLfo in the FIB is non-zero and any of the LFO entries referenced have a clfolvl of zero.

## Verification:

Modify an existing Word document to include an LFO entry with clfolvl set to zero and the reserved DWORD at offset eight to 0x41414141. Vulnerable applications crash when reading or closing the document.

## Identification:

Please see the Microsoft security bulletin for a list of affected files. The default location of WINWORD.EXE in Microsoft Word 2002 SP3 is "%ProgramFiles%\Microsoft Office\OFFICE10\".

## Tested Versions:

=====

The vulnerability was analysed on Windows XP SP2 with Microsoft Word 2002 (10.6846.6845) SP3 including WINWORD.EXE version 10.0.6846.0.

## Fixed Versions:

=====

The vulnerability is fixed in the patches released with MS08-072 by always setting the reserved value at offset eight in the LFO to zero before checking the value of clfolvl.

## References:

=====

## SA30285#1:

<http://secunia.com/advisories/30285>

## CVE-2008-4024:

[http://secunia.com/advisories/cve\\_reference/CVE-2008-4024/](http://secunia.com/advisories/cve_reference/CVE-2008-4024/)

## MS08-072 (KB957173):

<http://www.microsoft.com/technet/security/Bulletin/MS08-072.msp>

## Core Security Technologies:

<http://www.coresecurity.com/content/word-arbitrary-free>