



# SA30143 / CVE-2008-1091

**Generated by Secunia**

**16 May, 2008**

*5 pages*

## Table of Contents

<b>Introduction</b>	<b>2</b>
<b>Technical Details</b>	<b>2</b>
<b>Exploitation</b>	<b>4</b>
<b>Characteristics</b>	<b>4</b>
<b>Tested Versions</b>	<b>5</b>
<b>Fixed Versions</b>	<b>5</b>
<b>References</b>	<b>5</b>

## Introduction:

=====

A vulnerability in Microsoft Word when processing drawing objects in RTF files can be exploited by malicious people to compromise a user's system.

## Technical Details:

=====

Microsoft Word reads and writes "Rich Text Format" (RTF) as a file interchange format. The file specification is public and is based on text command words prefixed by backslashes ('\') and grouped by braces ('{' and '}'). It supports "Drawing Objects" that can use a number of drawing primitives such as lines and ellipses.

An integer overflow when parsing a large number of drawing primitives in a single drawing object can result in a heap-based buffer overflow.

While parsing the RTF file, if any drawing primitive command words are encountered ("dpgroup", "dpline", "dptxbx", "dprect", "dpellipse", "dparc", "dppolyline", "dpendgroup", "dpcallout", or "dppolygon") in a drawing object, then the function at sub\_305AD86E() will eventually be called to add the primitive to the current drawing group.

Previous processing will have initialised an object at offset 55B0h within the state object with the parameters of the command word, including a type ID found via lookup tables. The type ID is passed to sub\_30428D68() (not included in disassembly), which returns the number of bytes required to store it. The 16-bit size is saved at offset two in the sub-object.

```
.text:305AD86E sub_305AD86E  proc near                ; CODE XREF: sub_30589B88+2F1B
.text:305AD86E                                     ; sub_3058D8AA+483
.text:305AD86E
.text:305AD86E Obj                = byte ptr -60h
.text:305AD86E var_8              = dword ptr -8
.text:305AD86E var_4              = dword ptr -4
.text:305AD86E ppState           = dword ptr 8
.text:305AD86E
.text:305AD86E                push    ebp
.text:305AD86F                mov     ebp, esp
.text:305AD871                sub     esp, 60h
.text:305AD874                mov     eax, [ebp+ppState]
.text:305AD877                and     [ebp+var_4], 0
.text:305AD87B                push   ebx
.text:305AD87C                push   esi
.text:305AD87D                mov     esi, [eax]          ; pointer to state
.text:305AD87F                mov     ebx, [esi+5620h]    ; pointer to current drawing group object
.text:305AD885                push   edi
.text:305AD886                lea   edi, [esi+55B0h]     ; pointer to current command being processed
.text:305AD88C                mov   ax, [edi]           ; 16-bit RTF command word type id
.text:305AD88F                mov   [ebp+var_8], ebx
.text:305AD892                push   eax                ; uiType
.text:305AD893                call  sub_30428D68        ; get size of object of given type
.text:305AD898                mov   cx, [edi]
.text:305AD89B                cmp   cx, 806h
.text:305AD8A0                mov   [edi+2], ax        ; set size of object
.text:305AD8A4                jnz   short loc_305AD8B5 ; not dppolyline/dppolygon?
```

After some inconsequential operations, the current drawing group object pointer is checked.

```
.text:305AD907                test   ebx, ebx           ; drawing group object pointer
.text:305AD909                jnz   short loc_305AD953 ; drawing group object already exists?
```

If the pointer is null, a new object is allocated via MSO\_809() and uses the size from above (not shown in disassembly).

If, however, the drawing group object has already been allocated, then the size of the current array is added to the size of the primitive being processed. The new size is passed to MSO\_565() to resize the array to the new size.

```
.text:305AD953 loc_305AD953:                ; CODE XREF: sub_305AD86E+9B
.text:305AD953     mov     ax, [esi+5632h] ; ??
.text:305AD95A     test   al, 3
.text:305AD95C     jnz   short loc_305AD9A0 ; lower 2 bits set? jump not taken in testing
.text:305AD95E     mov     eax, [ebx] ; pointer to current array
.text:305AD960     movzx  eax, word ptr [eax+2] ; current size of array
.text:305AD964     movzx  ecx, word ptr [edi+2] ; size of new entry
.text:305AD968     add     eax, ecx ; new required size.
.text:305AD968     ; calculated as sum of two 16-bit values, but passed to
.text:305AD968     ; MSO_565() as the 32-bit sum.
.text:305AD96A     push   eax
.text:305AD96B     push   ebx
.text:305AD96C     call   MSO_565 ; reallocate?
.text:305AD972     test   eax, eax
.text:305AD974     jnz   short loc_305AD97D ; successful reallocation?
```

If the array was successfully resized, the size of the primitive being processed is added to the current size of the array stored at offset two within it. Note that the calculation is performed using 16-bit addition and may overflow.

```
.text:305AD97D loc_305AD97D:                ; CODE XREF: sub_305AD86E+106
.text:305AD97D     mov     eax, [ebp+ppState]
.text:305AD980     mov     esi, [eax] ; pointer to parser state
.text:305AD982     mov     eax, [ebx] ; pointer to reallocated array
.text:305AD984     lea   edi, [esi+55B0h] ; start of command being processed
.text:305AD98A     movzx  cx, [edi+2] ; size calculated above
.text:305AD98E     add     [eax+2], cx ; add the size of object to be appended to
.text:305AD98E     ; the total size of the array.
.text:305AD98E     ; Note: 16-bit addition may overflow
```

A pointer to the start of the newly allocated space in the array is calculated and the object representing the command being processed is copied into it by memmove().

```
.text:305AD992     movzx  ebx, word ptr [esi+562Ch] ; offset to end of array before being reallocated
.text:305AD999     mov     [ebp+ppState], eax ; overwrite with pointer to start of array
.text:305AD99C     add     ebx, eax ; compute pointer to newly allocated space in array
.text:305AD99E     jmp   short loc_305AD9CB
...
.text:305AD9CB loc_305AD9CB:                ; CODE XREF: sub_305AD86E+E3
.text:305AD9CB     ; sub_305AD86E+130
.text:305AD9CB     movzx  eax, word ptr [edi+2] ; size of newly processed command
.text:305AD9CF     push   eax ; size
.text:305AD9D0     mov     edx, ebx ; lpDest
.text:305AD9D2     mov     ecx, edi ; lpSrc
.text:305AD9D4     call   wrap_memmove ; write to newly allocated space in array
```

After some inconsequential operations, the offset to the end of the array is recalculated and saved in the state object. This is stored as a 16-bit value and there is no check to ensure the array has not grown beyond 64k bytes in which case the calculation will overflow.

```
.text:305AD9FE loc_305AD9FE:                ; CODE XREF: sub_305AD86E+174
.text:305AD9FE     ; sub_305AD86E+17B
.text:305AD9FE     mov     cx, [ebx+2] ; size of newly written object
.text:305ADA02     sub     cx, word ptr [ebp+ppState] ; treat pointer to start of array as 16-bit offset
.text:305ADA06     add     ecx, ebx ; size_used = (OldEndPointier-startPointer)+size
.text:305ADA08     mov     [esi+562Ch], cx ; offset to end of array, truncated to 16-bits
```

After some more inconsequential processing, the function returns and may be called again when another matching drawing primitive is encountered. If enough drawing primitives occur in a single drawing group, the 16-bit size and offset values will overflow.

While the overflowed size and offset does not cause a direct problem (they both overflow, therefore the new data is always written within the bounds of the allocated heap buffer), a heap-based buffer overflow can occur. The array has a 16-byte header followed by the actual drawing primitive entries. This is not seen in the processing described above since the offset (stored at 562Ch in the state object) always points to where the next entry should be written. However, when the offset overflows, it can result in a value less than 16, which causes the new entry to overwrite the array metadata.

#### Exploitation:

=====

By supplying a large number of drawing primitives in a single drawing group, the array metadata can be overwritten with controlled data (the parameters to the drawing primitive). The array metadata includes sizes and offsets that may lead to subsequent exploitable conditions.

Secunia Research has developed a PoC for the vulnerability. This is available to customers on Secunia Proof of Concept and Exploit Code Services.

#### Characteristics:

=====

#### Detection:

Look for an RTF document with a single drawing object containing a number of primitives where the sum of their sizes (see table below) plus 16 would exceed 65536.

primitive	size (in hex)
-----	
dpgroup	0e
dpline	26
dptxbx	28
dprect	26
dpellipse	24
dparc	26
dppolyline	32
dpendgroup	0e
dpcallout	7e
dppolygon	32

#### Verification:

Create an RTF document that consists solely of the "\rtf1" command word followed by 20000 "\\*\do" commands each containing a "\dpellipse" primitive. When opened in a vulnerable Word it will crash due to writing beyond the end of an allocated buffer. A debugger should be attached as some patched versions of Word will crash with a null pointer dereference due to a malformed RTF file.

## Identification:

Please see the Microsoft security bulletin for a list of affected versions. The default installation location of WORDVIEW.EXE in Microsoft Office Word Viewer SP3 is "%ProgramFiles%\Microsoft Office\OFFICE11\".

## Tested Versions:

=====

The vulnerability was analysed on Windows XP SP2 with Microsoft Office Word Viewer SP3 incorporating WORDVIEW.EXE version 11.0.8202.0.

## Fixed Versions:

=====

The vulnerability is patched by MS08-026 ignoring primitives if the array would exceed 64k bytes.

## References:

=====

### SA30143#1:

<http://secunia.com/advisories/30143/>

### CVE-2008-1091:

[http://secunia.com/cve\\_reference/CVE-2008-1091/](http://secunia.com/cve_reference/CVE-2008-1091/)

### MS08-026:

<http://www.microsoft.com/technet/security/bulletin/ms08-026.msp>

### ZDI:

<http://www.zerodayinitiative.com/advisories/ZDI-08-023/>