Secunia

Stay Secure

SA31675 / CVE-2008-3014

Generated by Secunia

10 September, 2008

5 pages

Table of Contents

Introduction:
==========

An integer overflow in Microsoft GDI+ during the processing of PolyPolygon records in WMF files can be exploited by malicious people to potentially compromise a user's system.

Technical Details:
==========

According to Microsoft, the Windows GDI+ "...provides two-dimensional vector graphics, imaging, and typography." Implemented in GDIPLUS.DLL, it contains support for parsing, among other formats, Windows Metafiles (WMF). Records in a WMF file are identified by 16-bit types documented by Microsoft.

An integer overflow when calculating the memory required for a PolyPolygon WMF record (type 0x0538) can be exploited to cause a heap-based buffer overflow and may potentially allow arbitrary code execution.

When parsing a WMF file with the GDI+ API, WmfEnumState::ProcessRecord() is called to process each record. WmfEnumState::PolyPolygon() is called in the case of record type 0x0538 (PolyPolygon).

```
...
.text:4ECD7C69 loc_4ECD7C69:                           ; CODE XREF: WmfEnumState::ProcessRecord(EmfPlusRecordType,uint,uchar const *)+BC
.text:4ECD7C69                 mov     ecx, 10538h
.text:4ECD7C6E                 cmp     eax, ecx
.text:4ECD7C70                 jg      loc_4ECD7D8B
.text:4ECD7C76                 jz      loc_4ECD7D7F
...
.text:4ECD7D7F loc_4ECD7D7F:                           ; CODE XREF: WmfEnumState::ProcessRecord(EmfPlusRecordType,uint,uchar const *)+288
.text:4ECD7D7F                 mov     ecx, esi
.text:4ECD7D81                 call    ?PolyPolygon@WmfEnumState@@QAEXXZ ; WmfEnumState::PolyPolygon(void)
```

In ::PolyPolygon(), a pointer to the content of the PolyPolygon record is retrieved and the first 16-bit parameter read, which indicates the number of polygons specified.

```
.text:4ECD5E2A ; public: void __thiscall WmfEnumState::PolyPolygon(void)
.text:4ECD5E2A ?PolyPolygon@WmfEnumState@@QAEXXZ proc near
.text:4ECD5E2A                                         ; CODE XREF: WmfEnumState::ProcessRecord(EmfPlusRecordType,uint,uchar const *)+393
.text:4ECD5E2A
.text:4ECD5E2A var_8           = dword ptr -8
.text:4ECD5E2A iTotalPoints    = dword ptr -4
.text:4ECD5E2A
.text:4ECD5E2A                 mov     edi, edi
.text:4ECD5E2C                 push    ebp
.text:4ECD5E2D                 mov     ebp, esp
.text:4ECD5E2F                 push    ecx
.text:4ECD5E30                 push    ecx
.text:4ECD5E31                 and     [ebp+iTotalPoints], 0
.text:4ECD5E35                 push    esi
.text:4ECD5E36                 mov     esi, ecx
.text:4ECD5E38                 mov     eax, [esi+5Ch]  ; pointer to record parameters
.text:4ECD5E3B                 push    edi
.text:4ECD5E3C                 movzx   edi, [eax+PolyPolygon.NumberOfPolygons]
.text:4ECD5E3F                 test    edi, edi
.text:4ECD5E41                 jbe     short loc_4ECD5E53 ; zero polygons?
```

If the number of polygons is non-zero, then a loop reads that number of 16-bit words each specifying the number of points in that polygon. A running total is kept of the total number of points in all the polygons.

```
.text:4ECD5E43                 add     eax, 2          ; skip to array aPointsPerPolygon
.text:4ECD5E46                 mov     ecx, edi
.text:4ECD5E48
.text:4ECD5E48 @loop_read_aPointsPerPolygon:           ; CODE XREF: WmfEnumState::PolyPolygon(void)+27
.text:4ECD5E48                 movzx   edx, word ptr [eax] ; PointsPerPolygon
.text:4ECD5E4B                 add     [ebp+iTotalPoints], edx
.text:4ECD5E4E                 inc     eax
.text:4ECD5E4F                 inc     eax
.text:4ECD5E50                 dec     ecx
.text:4ECD5E51                 jnz     short @loop_read_aPointsPerPolygon
```

The size required to store the points (two 32-bit values each) plus the array of polygon point counts (one 32-bit value each) is calculated. An integer overflow may occur in these operations, but there is no check for it. The resulting size is passed to ::CreateRecordToModify() to allocate a new blank record.

```
.text:4ECD5E53                 mov     eax, [ebp+iTotalPoints]
.text:4ECD5E56                 lea     eax, [edi+eax*2] ; num polygons + 2 * num points
.text:4ECD5E59                 shl     eax, 2          ; *4
.text:4ECD5E5C                 push    eax
.text:4ECD5E5D                 mov     ecx, esi
.text:4ECD5E5F                 call    ?CreateRecordToModify@MfEnumState@@IAEHH@Z ; MfEnumState::CreateRecordToModify(int)
```

::CreateRecordToModify() treats the size passed to it as a signed integer. If the value is negative, it instead uses the size of the WMF record from the record header.

```
.text:4ECD5406 ; protected: int __thiscall MfEnumState::CreateRecordToModify(int)
.text:4ECD5406 ?CreateRecordToModify@MfEnumState@@IAEHH@Z proc near
.text:4ECD5406                                         ; CODE XREF: EmfEnumState::CreateCopyOfCurrentRecord(void)+1D
.text:4ECD5406                                         ; EmfEnumState::CreateModifiedDib(tagBITMAPINFOHEADER *,uchar *,uint &,ulong)+85 ...
.text:4ECD5406
.text:4ECD5406 iSize           = dword ptr  8
.text:4ECD5406
.text:4ECD5406                 mov     edi, edi
.text:4ECD5408                 push    ebp
.text:4ECD5409                 mov     ebp, esp
.text:4ECD540B                 cmp     [ebp+iSize], 0
.text:4ECD540F                 push    esi
.text:4ECD5410                 mov     esi, ecx
.text:4ECD5412                 jg      short loc_4ECD541C ; positive, non-zero size?
.text:4ECD5414                 mov     eax, [esi]
.text:4ECD5416                 call    dword ptr [eax+0Ch] ; gdiplus.4ECD59D4 - WmfEnumState::GetCurrentRecordSize()
.text:4ECD5419                 mov     [ebp+iSize], eax ; use the size taken from the WMF record header
```

If the size is less than 2048 bytes, a pointer to a pre-allocated buffer is returned, otherwise a buffer is allocated using HeapAlloc() (not shown in disassembly). If the calculated requested size resulted in a negative value or the calculation overflowed, then an under-sized buffer may be allocated.

The WMF file data is then copied into the newly allocated structure. First, a loop reads the 16-bit point counts for each polygon, zero-extends it to 32 bits, and writes it to the new buffer. If an under-sized buffer was allocated, this may result in writing data past the end of the buffer.

```
.text:4ECD5E68                 mov     ecx, [esi+98h]  ; pointer to new Record
.text:4ECD5E6E                 lea     edx, [ecx+edi*4] ; pointer into new Record past aPointsPerPolygon
.text:4ECD5E71                 xor     eax, eax
.text:4ECD5E73                 test    edi, edi
.text:4ECD5E75                 push    ebx
.text:4ECD5E76                 mov     [ebp+lpPoints], edx
.text:4ECD5E79                 jbe     short loc_4ECD5E8B ; zero polygons?
.text:4ECD5E7B
.text:4ECD5E7B @loop_copy_aPointsPerPolygon:           ; CODE XREF: WmfEnumState::PolyPolygon(void)+5F
.text:4ECD5E7B                 mov     ebx, [esi+5Ch]  ; pointer to source record
.text:4ECD5E7E                 movzx   ebx, word ptr [ebx+eax*2+2] ; entry in aPointsPerPolygon array
.text:4ECD5E83                 mov     [ecx+eax*4], ebx ; write to destination record
.text:4ECD5E86                 inc     eax
.text:4ECD5E87                 cmp     eax, edi
.text:4ECD5E89                 jb      short @loop_copy_aPointsPerPolygon ; more polygons?
```

Next, another loop copies all the actual points into the new structure. Each point consists of two 16-bit values, which are sign-extended to 32-bits before writing. Again, if an under-sized buffer was allocated, this loop will overflow the buffer.

```
.text:4ECD5E8B                mov     ebx, [ebp+iTotalPoints]
.text:4ECD5E8E                test    ebx, ebx
.text:4ECD5E90                mov     eax, [esi+5Ch]  ; pointer to source record
.text:4ECD5E93                lea     eax, [eax+edi*2+2] ; start of points array
.text:4ECD5E97                jbe     short loc_4ECD5EB3 ; zero points?
.text:4ECD5E99                mov     [ebp+iTotalPoints], ebx
.text:4ECD5E9C
.text:4ECD5E9C @loop_copy_points:                     ; CODE XREF: WmfEnumState::PolyPolygon(void)+87
.text:4ECD5E9C                movsx   ebx, word ptr [eax]
.text:4ECD5E9F                mov     [edx], ebx
.text:4ECD5EA1                movsx   ebx, word ptr [eax+2]
.text:4ECD5EA5                mov     [edx+4], ebx
.text:4ECD5EA8                add     edx, 8          ; skip to next point in destination
.text:4ECD5EAB                add     eax, 4          ; skip to next point in source
.text:4ECD5EAE                dec     [ebp+iTotalPoints]
.text:4ECD5EB1                jnz     short @loop_copy_points ; more points?
```

Exploitation:
==========

The integer overflow can be triggered via a WMF file containing a specially crafted PolyPolygon record that specifies an overly large number of points. Attacker-controlled data will be written past the end of an under-sized heap buffer, ultimately triggering an access violation that will be handled by an exception handler.

The data written beyond the end of the allocated buffer is influenced by the attacker, but only the lower 16-bits of each 32-bit word can be controlled and the upper bits will be either all zeroes or all ones. As the attacker cannot specify a usable address, it appears unlikely that code execution would be possible, however, it cannot be completely ruled out.

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 a WMF file containing a PolyPolygon record (type 0x0538) where four times the number of polygons plus eight times the total number of points would exceed 0x7FFFFFFF.

Verification:
Create a WMF file containing a PolyPolygon record (type 0x0538) with a count of 8192 polygons each specifying 65535 points. When viewed by an application utilising a vulnerable version of GDI+, heap corruption will cause the process to terminate.

Identification:
Please see the Microsoft security bulletin for a list of affected versions. GDIPLUS.DLL is installed in a subfolder of "%WinDir%\WinSxS\".

Tested Versions:
==========

The vulnerability was analysed on Windows XP SP2 with GDIPLUS.DLL version 5.1.3102.2180.

Fixed Versions:
==========

The vulnerability is patched in the fixes released with MS08-52 by checking for overflows when calculating the required size.

References:
==========

SA31675:
http://secunia.com/advisories/31675/

CVE-2008-3014:
http://secunia.com/cve_reference/CVE-2008-3014/

MS08-052 (KB954593, KB938464):
http://www.microsoft.com/technet/security/Bulletin/MS08-052.mspx