TECHNICAL FEATURE

INSIDE THE WINDOWS META FILE FORMAT

Peter Ferrie
Symantec Security Response, USA

The Windows Meta File (WMF) format has received a lot of attention over recent weeks. In this article we will find out a bit more about it, and then discover why it has been in the spotlight.

PICTURE THIS

A metafile is a collection of records. Most commonly (although not always), these are used to describe a picture. The contents of the records correspond to particular graphics device interface (GDI) functions which, when 'played', will produce the image.

The minimum size for a metafile is 18 bytes. A file of this size would contain the header and no records.

The format of the file header is as follows:

Offset	Size	Description
00	2	type: 1 (memory) or 2 (disk)

(some documentation states, incorrectly, that 0 is a valid value)

Offset	Size	Description
02	2	number of words in header (must be 9)
04	2	version (0x100 or 0x300)
06	4	filesize in words
10	2	number of objects
12	4	maximum record size
16	2	number of parameters

Each record has the following format:

Offset	Size	Description
00	4	length of record
04	2	function number
06	n	record data

There is an extension to the WMF format, which is created by *Aldus*, and known as the 'placeable meta file'. The details of this format are not relevant here, except for the fact that a number of vulnerabilities in WMFs do not work if a placeable meta file is used. This is because the placeable meta file can only be used in a display device context, and cannot be printed.

Microsoft claims that the Escape function is disabled in placeable meta files, but in fact only certain subfunctions (most importantly, the SetAbortProc subfunction) are disabled. The relevance of this will become clear later.

STOP BUGGING ME

The final record in a WMF should be an EOF record. This is three words long, and its function number is zero. If the last record is not an EOF, *Windows* will parse the file searching for the EOF record. However, there are several bugs in the parsing process due to the fact that the file is assumed to be well formed.

First, if a zero-length record is encountered by *Windows* versions prior to *XP SP2*, the result is an infinite loop. This can be achieved with a 24-byte file. Although the bug was fixed in *Windows XP SP2*, it remains (at the time of writing) unpatched in previous versions of *Windows*, nearly two years after it was first disclosed.

The parser is supposed to scan the records from the start of the file to the end of the file, searching for the EOF record. However, since the values of the pointers are not checked in any way, the pointer to the next record may point backwards instead of forwards. It is possible for a backwards pointer to be followed by one or more forwards pointers, followed by another backwards pointer, and so on. Thus, it is vulnerable to circular linkages if a backwards pointer points to a list of forwards pointers that eventually point again to the same backwards pointer. All versions of *Windows*, including *XP SP2*, are vulnerable to this bug.

If the EOF record is found during the parsing, the in-memory copy of the file is truncated at that point, and the record count in the header is adjusted to account for the smaller size.

THE WHOLE HALF-TRUTH

The following is a list of the functions that, according to *Microsoft*, are the only functions supported by Windows Meta Files:

SetBkColor (1)	SetBkMode (2)
SetMapMode (3)	SetROP2 (4)
SetPolyFillMode (6)	SetStretchBltMode (7)
SetTextCharacterExtra (8)	SetTextColor (9)
SetTextJustification (10)	SetWindowOrgEx (11)
SetWindowExtEx (12)	SetViewportOrgEx (13)
SetViewportExtEx (14)	OffsetWindowOrgEx (15)
ScaleWindowExtEx (16)	OffsetViewportOrgEx (17)

ScaleViewportExtEx (18)	LineTo (19)
MoveToEx (20)	ExcludeClipRect (21)
IntersectClipRect (22)	Arc (23)
Ellipse (24)	FloodFill (25)
Pie (26)	Rectangle (27)
RoundRect (28)	PatBlt (29)
SaveDC (30)	SetPixel (31)
OffsetClipRgn (32)	TextOutA (33)
BitBlt (34)	StretchBlt (35)
Polygon (36)	Polyline (37)
Escape (38)	RestoreDC (39)
FillRgn (40)	FrameRgn (41)
InvertRgn (42)	PaintRgn (43)
SelectClipRgn (44)	SelectObject (45)
SetTextAlign (46)	Chord (48)
SetMapperFlags (49)	ExtTextOutA (50)
SetDIBitsToDevice (51)	SelectPalette (52)
RealizePalette (53)	AnimatePalette (54)
SetPaletteEntries (55)	PolyPolygon (56)
ResizePalette (57)	CreateDIBPatternBrush (66)
StretchDIBits (67)	ExtFloodFill (72)
DeleteObject (240)	CreatePalette (247)
CreatePatternBrush (249)	CreatePenIndirect (250)
CreateFontIndirect (251)	CreateBrushIndirect (252)

The truth is a little different however. We find that the DIBBitBlt (64) and DIBStretchBlt (65) functions exist, but are not listed. The CreateRectRgn (255) function is not listed either, but this exists in all versions of *Windows* including *Windows 3.x*. Finally, the SetLayout (73) function is not listed, but exists in *Windows 2000* and later.

SOMETHING LIKE THAT

As is often the case with unusual file formats, when an exploit appears, bad documentation follows it. In this case, there were descriptions of which of the fields were meaningful, and which were not.

While some of the documentation was correct (for example, the upper byte of the WMF function number is not checked, it serves merely as a hint to the number of parameters that are expected to be passed), some of it was not. For example, the 'number of objects' field was documented as being unnecessary, when in fact a valid value is required by the *Rgn functions and by SelectObject.

IN ... SECURE

Security seems not to have been a prime consideration when the WMF format was first introduced, and the programmer of the parser was incredibly trusting. As several of us found, a total of eight functions were vulnerable to 15 different buffer overflow conditions that could allow remote code execution. This prompted *Microsoft* to release security bulletin MS05-053. The vulnerable functions were:

AnimatePalette	SetPaletteEntries
PolyPolygon	DIBBitBlt
DIBStretchBlt	CreateDIBPatternBrush
CreatePalette	CreatePatternBrush

In fact, the CreateRectRgn function was also vulnerable, but an attack against this would require a file that was one gigabyte in size.

In addition to the denial-of-service attacks described above, at least 14 functions are known to be vulnerable to conditions that cause *Internet Explorer* on all platforms, and *Windows Explorer* on *Windows XP* (including *SP2*), to crash instantly upon opening malformed files. The vulnerable functions are the same as those listed for the buffer overflow functions above, including the CreateRectRgn function, with the addition of the following functions:

SetBkMode	TextOutA
BitBlt	StretchBlt
ExtTextOutA	SetDIBitsToDevice
StretchDIBits	

[W]ANT [M]ORE [F]REEDOM

One function is of particular interest in WMF format: the Escape function. The Escape function enables applications to bypass the GDI layer, and communicate directly with a particular device. This communication is intended to be directed to a printer, but the display device will accept some of the commands too.

The Escape function supports a number of subfunctions, most of which are related to printer control, such as StartDoc and StartPage, and the corresponding EndDoc and EndPage. Not surprisingly, at least three of these subfunctions contain bugs.

The bugs appear if a non-placeable WMF calls the StartDoc (3 or 4110) or StartPage (10) subfunction before any call is made to CreateDC(). This is possible in *Windows Explorer* on *Windows XP*, for example, because there the created device context is compatible with both printer and display devices. The result is that the viewing application will crash. In order to attack the *Windows XP* platform, where the GDI+ layer exists, the minimum file length is 62 bytes.

Finally, we reach the most trusting part of the WMF format parser, which is the cause of most of the trouble: the SetAbortProc subfunction.

SETABORTPROC

The SetAbortProc function has existed since the days before *Windows 3.0*. That's over 15 years! It was implemented in the days of cooperative multi-tasking – before there were threads – in which an application was required to yield CPU control explicitly to other applications.

The function was designed to allow an application to cancel a print job once it had started, and the only way in which that could happen was through the use of a callback function that was called periodically. This was fine until the WMF format was introduced and the abort functionality was added to it. At that point, the WMF itself could carry its own abort handler. An image file containing executable code? It's unthinkable today, but that was then, this is now.

The Escape record subfunctions exist as part of the standard record data:

Offset	Size	Description
00	2	subfunction number
(all 16 bits are checked here)		
02	2	size of input structure
04	n	input structure

The SetAbortProc subfunction number has a value of 9, the value in the 'size of input structure' field is ignored, and the 'input structure' is the handler code.

While only one function handler can be registered at any one time, the SetAbortProc function can be called multiple times from within a WMF, so it is possible to register different handlers at different times during the parsing of the file. This allows for a variety of effects, and could have been used for a multi-stage attack, which would potentially have been difficult to detect.

Once the function handler is registered, it is called before each of the following records is parsed. Although the function is documented as being used to abort the printing of the image, an undocumented side effect is that it can also be used to abort the rendering of the image. It is not clear whether this particular behaviour is intentional, but if it is, that would explain why a device context does not have to refer only to a printer.

The function handler accepts two parameters. This leads to another bug: *Windows* does not check that those parameters are removed from the stack when the handler returns, so a sufficiently large (or circularly-linked) WMF can exhaust the stack space and cause a stack fault. If *Windows Explorer* attempts to display such a file, *Explorer* exits silently and suddenly, and no error message is displayed.

SERVICE D.E.P.ARTMENT

Windows XP SP2 introduced the Data Execution Prevention technology, which prevents pages that are marked as data from executing code. Its primary goal is to make it harder for buffer overflows to gain control of the CPU. A side effect is that it also stops the SetAbortProc function handler from executing, since the GDI does not mark the pages as executable.

AREA 51

Security researcher/commentator Steve Gibson has recently aired some controversial opinions on the WMF vulnerabilities, suggesting that they may not, in fact, have been accidental. However, in this case, his examination of 'exactly how it works' proved to be about as incorrect as it can get.

He claimed that the attack worked on *Windows 2000*. Presumably, that was by playing it through a dedicated application, since there is no default handler for WMFs on that platform, and *Internet Explorer* plays only placeable meta files which, as mentioned before, will not run the SetAbortProc function.

Gibson claimed that the record length must be set to 1 in order to run the code. This is untrue. The record length can be any value, as long as it remains within the bounds of the file *and* the next record function is not EOF. This last part is critical. The function is called only when the next record is reached, but processing stops when EOF is encountered. Thus, if the WMF contains only SetAbortProc and EOF, then only a malformed record length will point to something that remains within the file but does not point to an EOF record.

The reason why a value of 2 would not work is that the 'function number' field in the next record corresponds to the 'size of input structure' field in the SetAbortProc record. If the input size is set to zero, it will look like EOF.

The reason why a record length of 0 does not work is related to the zero-length bug described above. That bug actually exists in two locations – one when parsing the file to find the EOF record, and one while parsing the file in order to render it. While the first case was fixed only in *Windows XP SP2*, the second bug was fixed in *Windows 2000* too. The second fix is relatively recent, though, since a default *Windows XP SP1* installation, for example, is vulnerable.

Gibson claimed that a thread is created to run the SetAbortProc handler. In fact, no thread is created to run the handler – it is a callback, which is called by the parser, and the parser has to wait until the callback returns, otherwise the whole point of the function (to abort the printing) is lost.

By his own admission, Gibson did not read the documentation (in fact, he claimed that he couldn't find it, although it is freely available on *Microsoft*'s website), and he claimed that the device context is not available to the function handler. Of course the device context is available to the function handler – it is one of the two parameters that is passed to it (see above), and it is required in order to abort the printing.

Finally, Gibson claimed that the control flow could not return to *Windows*. It is simply a matter of the function returning and discarding the parameters that were passed on the stack. If the record is well formed, *Windows* will continue to parse the file, as before.

I GUESS ...

Gibson admits that he was guessing about a number of things. Unfortunately, he guessed poorly. I guess we know better now.

CONCLUSION

So what are the consequences of the WMF bug and who really is vulnerable? It all comes down to the software that is installed on the machine.

Machines running *Windows XP* are vulnerable without user interaction, because *XP* has a default handler for WMFs that can be launched from within *Internet Explorer* without user interaction. Email programs, such as *Microsoft Outlook*, which support the display of media through an IFrame, are also a vector for system compromise when previewing or opening an email.

Earlier platforms, such as *Windows 9x*, *NT*, and *2000*, all contain the same vulnerability, but without a default handler they cannot be exploited in the same way. However, anyone using those platforms who has installed software that handles WMFs will be vulnerable to the same kind of attacks.