# Windows Metafiles

## An Analysis of the EMF Attack Surface & Recent Vulnerabilities

Mateusz “j00ru” Jurczyk

PacSec, Tokyo 2016

## PS> whoami

- Project Zero @ Google
- Low-level security researcher with interest in all sorts of vulnerability research and software exploitation
- http://j00ru.vexillium.org/
- @j00ru

## Agenda

- Windows Metafile primer, GDI design, attack vectors.
- Hacking:
  - Internet Explorer (GDI)
  - Windows Kernel (ATMFD.DLL)
  - Microsoft Office (GDI+)
  - VMware virtualization (Print Spooling)
- Final thoughts.

## Windows GDI & Metafile primer

## Windows GDI

- GDI stands for *Graphics Device Interface*.
- Enables user-mode applications to use graphics and formatted text on video displays and printers.
- Major part of the system API (nearly 300 documented functions).
- Present in the OS since the very beginning (Windows 1.0 released in 1985).
  - One of the oldest subsystems, with most of its original code still running 31 years later.
  - Concidentally (?) also one of the most buggy components.

## How to draw

1. Grab a handle to a Device Context (HDC).
    - Identifies a persistent container of various graphical settings (pens, brushes, palettes etc.).
    - Can be used to draw to a screen (most typically), a printer, or a metafile.
    - Most trivial example:

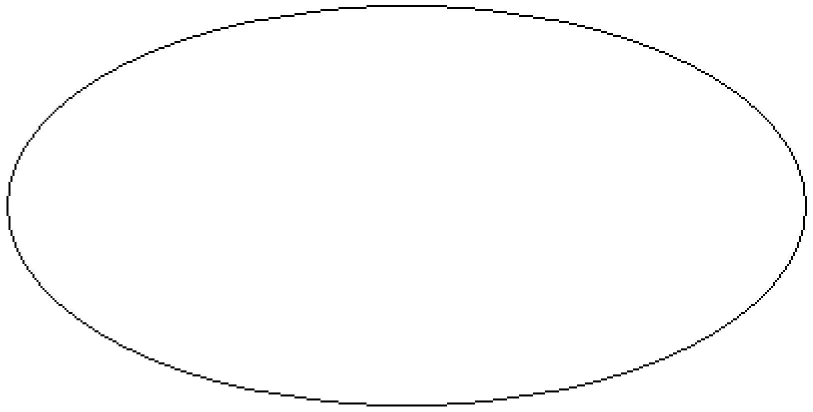
```
HDC hdc = GetDC(NULL);
```

*(obtains a HDC for the entire screen)*

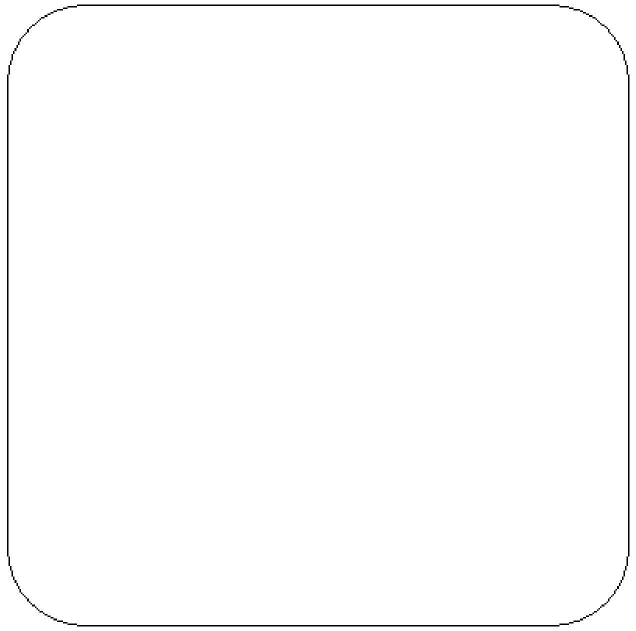
# How to draw

2. Use a drawing function.

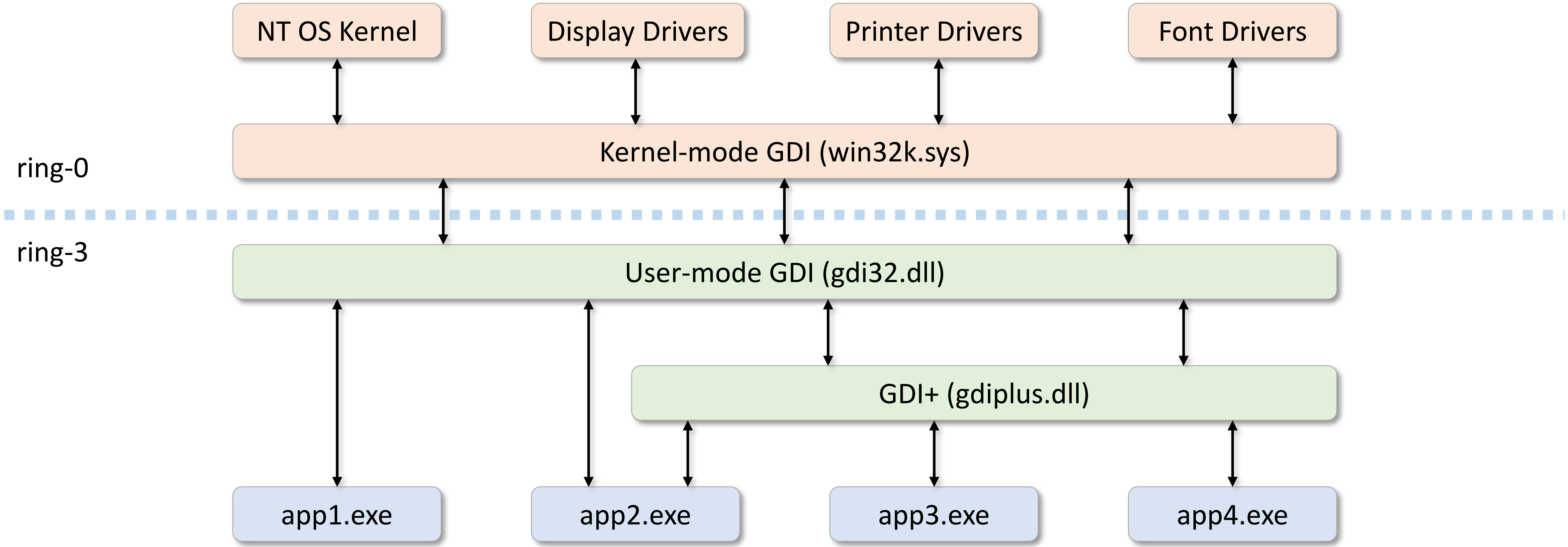
```
Ellipse(hdc, 100, 100, 500, 300);
```



```
RoundRect(hdc, 100, 100, 500, 500, 100, 100);
```



# Windows GDI – simplified architecture



## User to kernel API mappings

Most user-mode GDI functions have their direct counterparts in the kernel:

| GDI32.DLL | win32k.sys |
|---|---|
| `AbortDoc` | `NtGdiAbortDoc` |
| `AbortPath` | `NtGdiAbortPath` |
| `AddFontMemResourceEx` | `NtGdiAddFontMemResourceEx` |
| `AddFontResourceW` | `NtGdiAddFontResourceW` |
| `AlphaBlend` | `NtGdiAlphaBlend` |
| ... | ... |

## Windows Metafiles

Core idea:

Store images as lists of records directly describing GDI calls.

## Windows Metafiles

- **Pros:**
  - requires little computation work from the rasterizer itself, as it only has to call GDI functions with the supplied parameters.
  - provides an official way of serializing sets of GDI operations into reproducible images.
  - can work as a vector format, raster, or both.
- **Cons:**
  - only works on Windows, unless full implementation of the supported graphical GDI operations is implemented externally.

## First version: WMF

- The original metafiles (WMF = **W**indows **M**eta**F**iles).
- Introduced with Windows 3.0 in 1990.
  - Not as ancient as GDI itself, but almost so.
- Initially documented in Windows 3.1 SDK (1994, volume 4).
  - A revised, more complete specification was released in 2006, and has been maintained since then.
  - A description of all records and structures can be found in the MS-WMF document.

# WMF files – 60 supported API functions

| | | |
|---|---|---|
| `AnimatePaletteArc` | `LineToMoveToEx` | `SelectPaletteSetBkColor` |
| `BitBlt` | `OffsetClipRgn` | `SetBkMode` |
| `Chord` | `OffsetViewportOrgEx` | `SetDIBitsToDevice` |
| `CreateBrushIndirect` | `OffsetWindowOrgEx` | `SetMapMode` |
| `CreateDIBPatternBrush` | `PaintRgn` | `SetMapperFlags` |
| `CreateFontIndirect` | `PatBlt` | `SetPaletteEntries` |
| `CreatePalette` | `Pie` | `SetPixel` |
| `CreatePatternBrush` | `Polygon` | `SetPolyFillMode` |
| `CreatePenIndirect` | `Polyline` | `SetROP2` |
| `DeleteObject` | `PolyPolygon` | `SetStretchBltMode` |
| `Ellipse` | `RealizePalette` | `SetTextAlign` |
| `Escape` | `Rectangle` | `SetTextCharacterExtra` |
| `ExcludeClipRect` | `ResizePalette` | `SetTextColor` |
| `ExtFloodFill` | `RestoreDC` | `SetTextJustification` |
| `ExtTextOut` | `RoundRect` | `SetViewportOrgEx` |
| `FillRgn` | `SaveDC` | `SetWindowExtEx` |
| `FloodFill` | `ScaleViewportExtEx` | `SetWindowOrgEx` |
| `FrameRgn` | `ScaleWindowExtEx` | `StretchBlt` |
| `IntersectClipRect` | `SelectClipRgn` | `StretchDIBits` |
| `InvertRgn` | `SelectObject` | `TextOut` |

# Some seemingly interesting ones

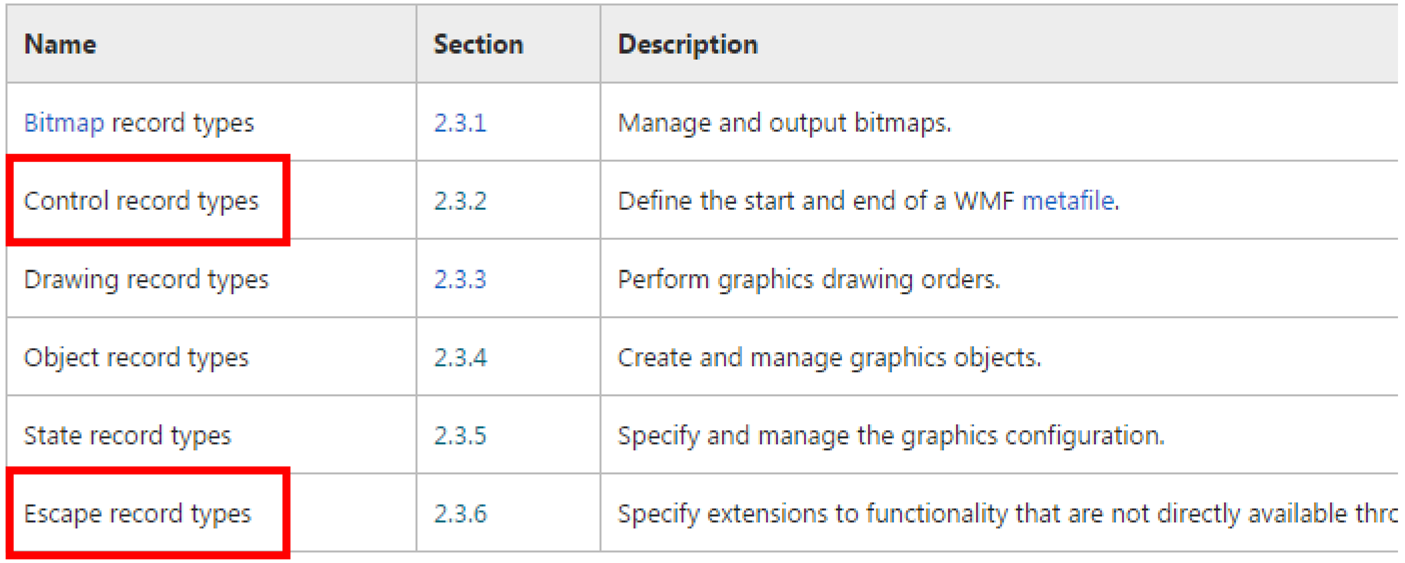
| | | |
|---|---|---|
| AnimatePaletteArc | LineToMoveToEx | SelectPaletteSetBkColor |
| BitBlt | OffsetClipRgn | SetBkMode |
| Chord | OffsetViewportOrgEx | SetDIBitsToDevice |
| CreateBrushIndirect | OffsetWindowOrgEx | SetMapMode |
| CreateDIBPatternBrush | PaintRgn | SetMapperFlags |
| CreateFontIndirect | PatBlt | SetPaletteEntries |
| CreatePalette | Pie | SetPixel |
| CreatePatternBrush | Polygon | SetPolyFillMode |
| CreatePenIndirect | Polyline | SetROP2 |
| **DeleteObject** | PolyPolygon | SetStretchBltMode |
| Ellipse | RealizePalette | SetTextAlign |
| **Escape** | Rectangle | SetTextCharacterExtra |
| ExcludeClipRect | ResizePalette | SetTextColor |
| ExtFloodFill | **RestoreDC** | SetTextJustification |
| ExtTextOut | RoundRect | SetViewportOrgEx |
| FillRgn | **SaveDC** | SetWindowExtEx |
| FloodFill | ScaleViewportExtEx | SetWindowOrgEx |
| FrameRgn | ScaleWindowExtEx | StretchBlt |
| IntersectClipRect | SelectClipRgn | StretchDIBits |
| InvertRgn | **SelectObject** | TextOut |

## WMF: there’s more!

- The format also supports a number of records which do not directly correspond to GDI functions.
  - Header with metadata.
  - Embedded EMF.
  - Records directly interacting with the printer driver / output device.
  - End-of-file marker.
  - ...

# WMF: there's more!

- Generally, the most interesting records can be found in two sections:



| Name | Section | Description |
|---|---|---|
| Bitmap record types | 2.3.1 | Manage and output bitmaps. |
| Control record types | 2.3.2 | Define the start and end of a WMF metafile. |
| Drawing record types | 2.3.3 | Perform graphics drawing orders. |
| Object record types | 2.3.4 | Create and manage graphics objects. |
| State record types | 2.3.5 | Specify and manage the graphics configuration. |
| Escape record types | 2.3.6 | Specify extensions to functionality that are not directly available thro |

# Windows Metafile – example

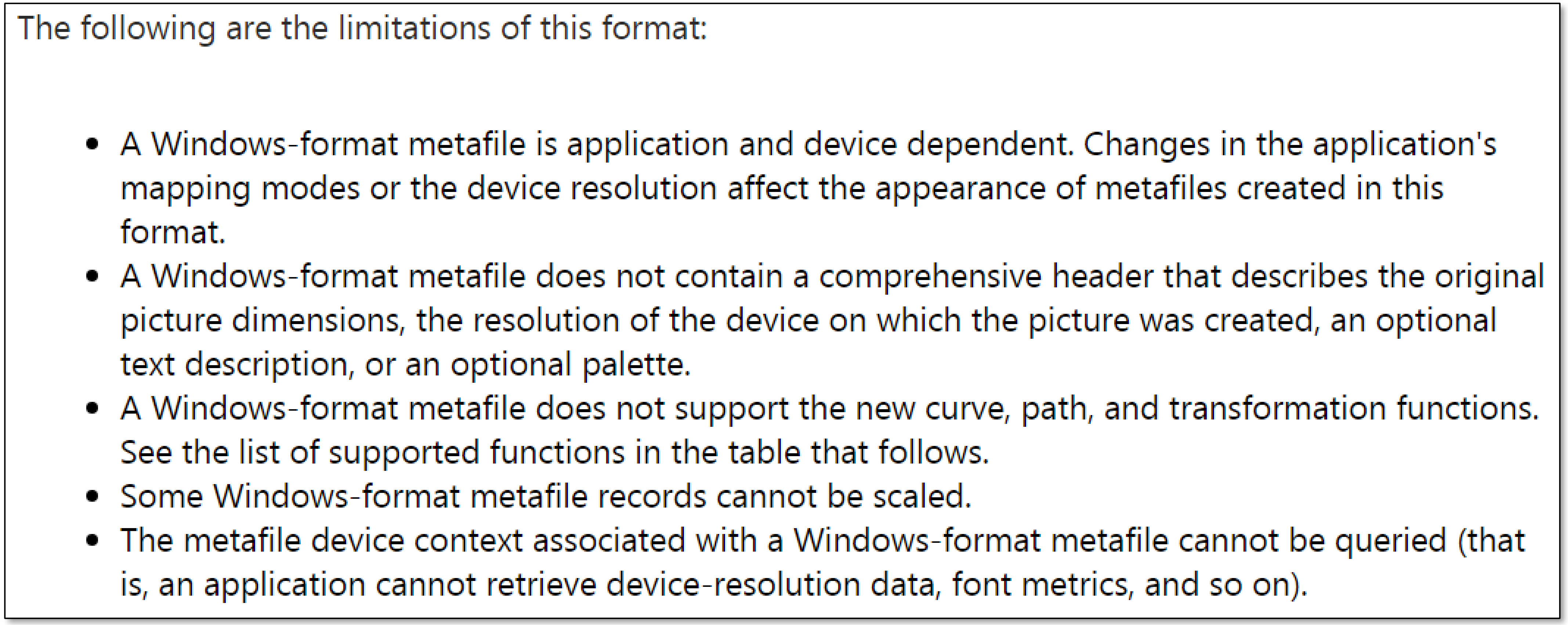
```
...
R0003: [017] META_SETMAPMODE           (s=12) {iMode(8=MM_ANISOTROPIC)}
R0004: [011] META_SETVIEWPORTEXTEX     (s=16) {szlExtent(1920,1200)}
R0005: [009] META_SETWINDOWEXTEX       (s=16) {szlExtent(1920,1200)}
R0006: [010] META_SETWINDOWORGEX       (s=16) {ptlOrigin(-3972,4230)}
R0007: [009] META_SETWINDOWEXTEX       (s=16) {szlExtent(7921,-8462)}
R0008: [049] META_CREATEPALETTE        (s=960) {ihPal(1) LOGPAL[ver:768, entries:236]}
R0009: [048] META_SELECTPALETTE        (s=12) {ihPal(Table object: 1)}
R0010: [052] META_REALIZEPALETTE       (s=8)
R0011: [039] META_CREATEBRUSHINDIRECT  (s=24) {ihBrush(2), style(0=BS_SOLID, color:0x00FFFFFF)}
R0012: [037] META_SELECTOBJECT         (s=12) {Table object: 2=OBJ_BRUSH.(BS_SOLID)}
R0013: [037] META_SELECTOBJECT         (s=12) {Stock object: 8=OBJ_PEN.(PS_NULL)}
R0014: [019] META_SETPOLYFILLMODE      (s=12) {iMode(1=ALTERNATE)}
R0015: [086] META_POLYGON16            (s=320) {rclBounds(89,443,237,548), nbPoints:73, P1(-2993,398) - Pn(-2993,398)}
R0016: [038] META_CREATEPEN            (s=28) {ihPen(3), style(0=PS_SOLID | COSMETIC), width(0), color(0x00000000)}
...
```

## WMF: still very obsolete

- Even though already quite complex, the format didn’t turn out to be very well thought-out for modern usage.
- It’s still supported by GDI, and therefore some of its clients (e.g. Microsoft Office, Paint, some default Windows apps).
- Has been basically forgotten in any real-world use-cases for the last decade or more.

# WMF: discouraged from use

- Even Microsoft gives a lot of reasons not to use it anymore:



The following are the limitations of this format:

- A Windows-format metafile is application and device dependent. Changes in the application's mapping modes or the device resolution affect the appearance of metafiles created in this format.
- A Windows-format metafile does not contain a comprehensive header that describes the original picture dimensions, the resolution of the device on which the picture was created, an optional text description, or an optional palette.
- A Windows-format metafile does not support the new curve, path, and transformation functions. See the list of supported functions in the table that follows.
- Some Windows-format metafile records cannot be scaled.
- The metafile device context associated with a Windows-format metafile cannot be queried (that is, an application cannot retrieve device-resolution data, font metrics, and so on).

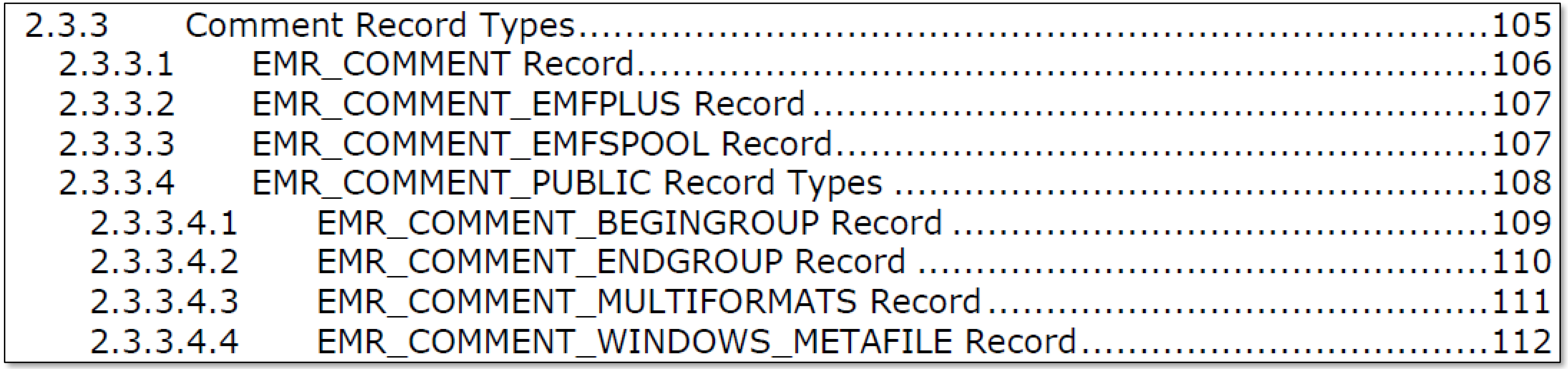
## Next up: EMF (**E**nhanced **M**eta**F**iles)

- Already in 1993, Microsoft released an improved revision of the image format, called EMF.
- Documented in the official MS-EMF specification.
- Surpasses WMF in a multitude of ways:
  - uses 32-bit data/offset width, as opposed to just 16 bits.
  - device independent.
  - supports a number of new GDI calls, while maintaining backward compatibility with old records.

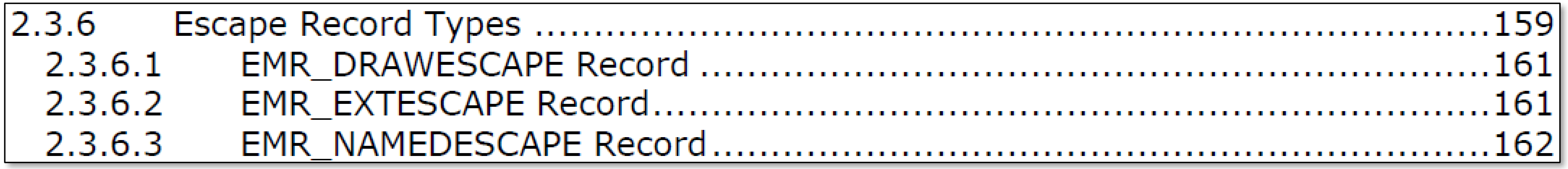
# Enhanced Metafile – example

```
...
R0121: [039] EMR_CREATEBRUSHINDIRECT  (s=24) {ihBrush(2), style(1=BS_NULL)}
R0122: [037] EMR_SELECTOBJECT         (s=12) {Table object: 2=OBJ_BRUSH.(BS_NULL)}
R0123: [040] EMR_DELETEOBJECT         (s=12) {ihObject(1)}
R0124: [090] EMR_POLYPOLYLINE16       (s=44) {rclBounds(128,-256,130,-254), nPolys:1, nbPoints:2, P1(386,-765) - Pn(386,-
765)}
R0125: [019] EMR_SETPOLYFILLMODE      (s=12) {iMode(1=ALTERNATE)}
R0126: [039] EMR_CREATEBRUSHINDIRECT  (s=24) {ihBrush(1), style(0=BS_SOLID, color:0x00A86508)}
R0127: [037] EMR_SELECTOBJECT         (s=12) {Table object: 1=OBJ_BRUSH.(BS_SOLID)}
R0128: [040] EMR_DELETEOBJECT         (s=12) {ihObject(2)}
R0129: [058] EMR_SETMITERLIMIT        (s=12) {Limit:0.000}
R0130: [091] EMR_POLYPOLYGON16        (s=60) {rclBounds(127,-259,138,-251), nPolys:1, nbPoints:6, P1(384,-765) - Pn(384,-
765)}
R0131: [040] EMR_DELETEOBJECT         (s=12) {ihObject(1)}
R0132: [040] EMR_DELETEOBJECT         (s=12) {ihObject(3)}
R0133: [014] EMR_EOF                  (s=20) {nPalEntries:0, offPalEntries:16, nSizeLast:20}
...
```

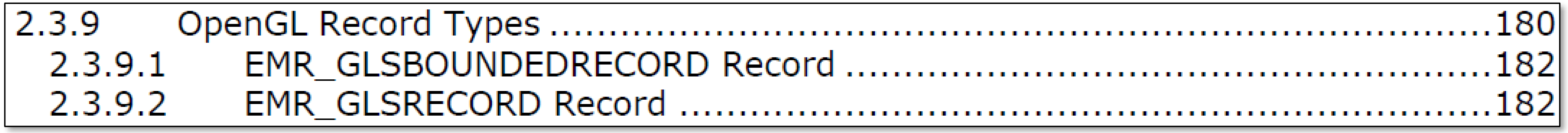
## EMF: interesting records at first glance



# EMF: interesting records at first glance



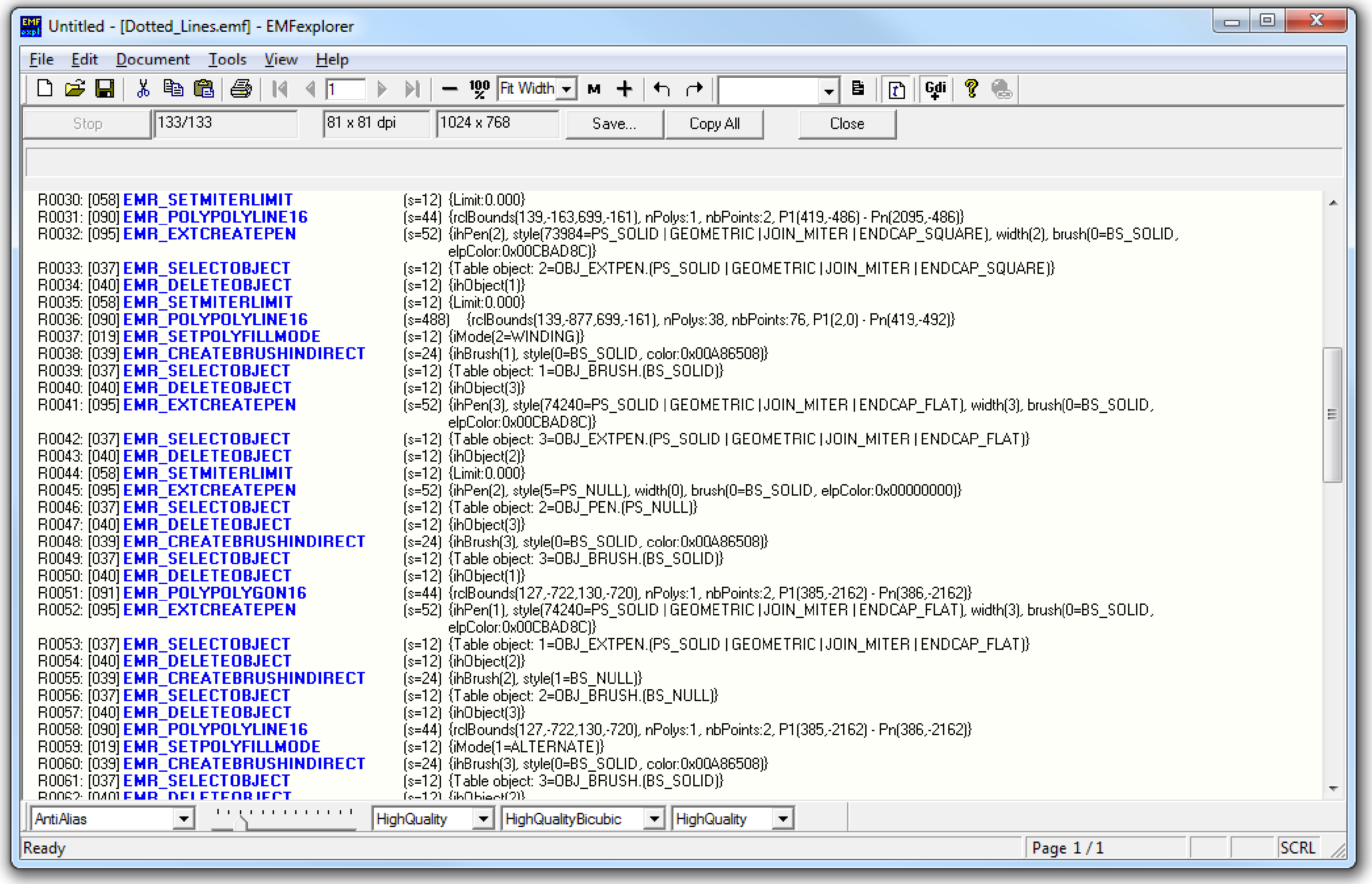
# EMF: interesting records at first glance



## EMF: current support

- Despite being only 3 years younger than WMF, EMF has remained in current usage until today.
    - Not as a mainstream image format, but still a valid attack vector.
- A variety of attack vectors:
    - Win32 GDI clients – most notably Internet Explorer.
    - GDI+ clients – most notably Microsoft Office.
    - Printer drivers, including those used in virtualization technology.

# Toolset – examination (EMFexplorer)

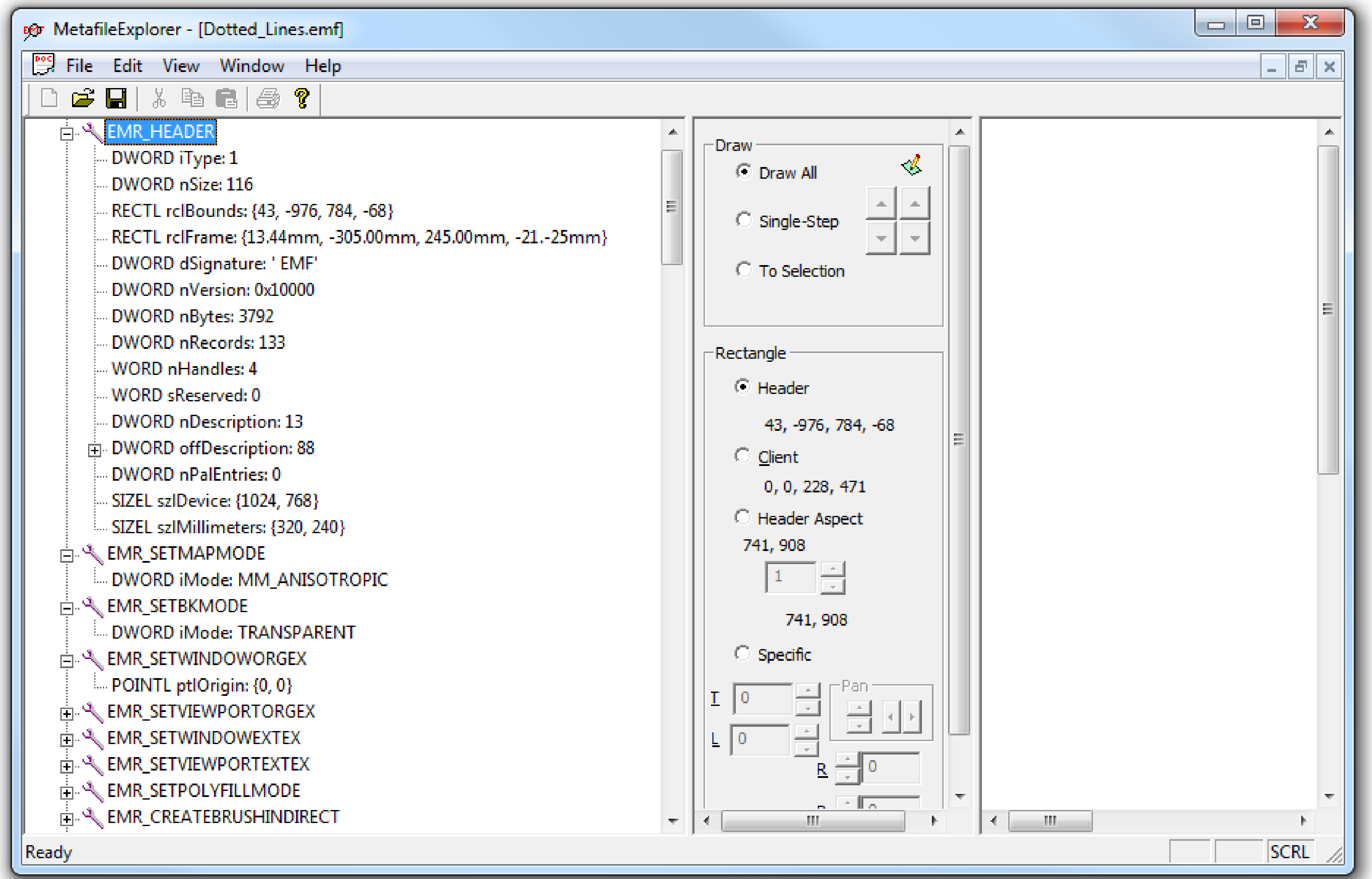

Untitled - [Dotted_Lines.emf] - EMFexplorer
File Edit Document Tools View Help
Fit Width
Stop
133/133
81 x 81 dpi
1024 x 768
Save...
Copy All
Close

```
R0030: [058] EMR_SETMITERLIMIT          (s=12) {Limit:0.000}
R0031: [090] EMR_POLYPOLYLINE16         (s=44) {rclBounds(139,-163,699,-161), nPolys:1, nbPoints:2, P1(419,-486) - Pn(2095,-486)}
R0032: [095] EMR_EXTCREATEPEN           (s=52) {ihPen(2), style(73984=PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_SQUARE), width(2), brush(0=BS_SOLID,
                                               elpColor:0x00CBAD8C)}
R0033: [037] EMR_SELECTOBJECT           (s=12) {Table object: 2=OBJ_EXTPEN.(PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_SQUARE)}
R0034: [040] EMR_DELETEOBJECT           (s=12) {ihObject(1)}
R0035: [058] EMR_SETMITERLIMIT          (s=12) {Limit:0.000}
R0036: [090] EMR_POLYPOLYLINE16         (s=488)   {rclBounds(139,-877,699,-161), nPolys:38, nbPoints:76, P1(2,0) - Pn(419,-492)}
R0037: [019] EMR_SETPOLYFILLMODE        (s=12) {iMode(2=WINDING)}
R0038: [039] EMR_CREATEBRUSHINDIRECT    (s=24) {ihBrush(1), style(0=BS_SOLID, color:0x00A86508)}
R0039: [037] EMR_SELECTOBJECT           (s=12) {Table object: 1=OBJ_BRUSH.(BS_SOLID)}
R0040: [040] EMR_DELETEOBJECT           (s=12) {ihObject(3)}
R0041: [095] EMR_EXTCREATEPEN           (s=52) {ihPen(3), style(74240=PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_FLAT), width(3), brush(0=BS_SOLID,
                                               elpColor:0x00CBAD8C)}
R0042: [037] EMR_SELECTOBJECT           (s=12) {Table object: 3=OBJ_EXTPEN.(PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_FLAT)}
R0043: [040] EMR_DELETEOBJECT           (s=12) {ihObject(2)}
R0044: [058] EMR_SETMITERLIMIT          (s=12) {Limit:0.000}
R0045: [095] EMR_EXTCREATEPEN           (s=52) {ihPen(2), style(5=PS_NULL), width(0), brush(0=BS_SOLID, elpColor:0x00000000)}
R0046: [037] EMR_SELECTOBJECT           (s=12) {Table object: 2=OBJ_PEN.(PS_NULL)}
R0047: [040] EMR_DELETEOBJECT           (s=12) {ihObject(3)}
R0048: [039] EMR_CREATEBRUSHINDIRECT    (s=24) {ihBrush(3), style(0=BS_SOLID, color:0x00A86508)}
R0049: [037] EMR_SELECTOBJECT           (s=12) {Table object: 3=OBJ_BRUSH.(BS_SOLID)}
R0050: [040] EMR_DELETEOBJECT           (s=12) {ihObject(1)}
R0051: [091] EMR_POLYPOLYGON16          (s=44) {rclBounds(127,-722,130,-720), nPolys:1, nbPoints:2, P1(385,-2162) - Pn(386,-2162)}
R0052: [095] EMR_EXTCREATEPEN           (s=52) {ihPen(1), style(74240=PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_FLAT), width(3), brush(0=BS_SOLID,
                                               elpColor:0x00CBAD8C)}
R0053: [037] EMR_SELECTOBJECT           (s=12) {Table object: 1=OBJ_EXTPEN.(PS_SOLID | GEOMETRIC | JOIN_MITER | ENDCAP_FLAT)}
R0054: [040] EMR_DELETEOBJECT           (s=12) {ihObject(2)}
R0055: [039] EMR_CREATEBRUSHINDIRECT    (s=24) {ihBrush(2), style(1=BS_NULL)}
R0056: [037] EMR_SELECTOBJECT           (s=12) {Table object: 2=OBJ_BRUSH.(BS_NULL)}
R0057: [040] EMR_DELETEOBJECT           (s=12) {ihObject(3)}
R0058: [090] EMR_POLYPOLYLINE16         (s=44) {rclBounds(127,-722,130,-720), nPolys:1, nbPoints:2, P1(385,-2162) - Pn(386,-2162)}
R0059: [019] EMR_SETPOLYFILLMODE        (s=12) {iMode(1=ALTERNATE)}
R0060: [039] EMR_CREATEBRUSHINDIRECT    (s=24) {ihBrush(3), style(0=BS_SOLID, color:0x00A86508)}
R0061: [037] EMR_SELECTOBJECT           (s=12) {Table object: 3=OBJ_BRUSH.(BS_SOLID)}
R0062: [040] EMR_DELETEOBJECT           (s=12) {ihObject(2)}
```

AntiAlias | HighQuality | HighQualityBicubic | HighQuality

Ready | Page 1 / 1 | SCRL

# Toolset – examination (MetafileExplorer)



```
EMR_HEADER
  DWORD iType: 1
  DWORD nSize: 116
  RECTL rclBounds: {43, -976, 784, -68}
  RECTL rclFrame: {13.44mm, -305.00mm, 245.00mm, -21.-25mm}
  DWORD dSignature: ' EMF'
  DWORD nVersion: 0x10000
  DWORD nBytes: 3792
  DWORD nRecords: 133
  WORD nHandles: 4
  WORD sReserved: 0
  DWORD nDescription: 13
  DWORD offDescription: 88
  DWORD nPalEntries: 0
  SIZEL szlDevice: {1024, 768}
  SIZEL szlMillimeters: {320, 240}
EMR_SETMAPMODE
  DWORD iMode: MM_ANISOTROPIC
EMR_SETBKMODE
  DWORD iMode: TRANSPARENT
EMR_SETWINDOWORGEX
  POINTL ptlOrigin: {0, 0}
EMR_SETVIEWPORTORGEX
EMR_SETWINDOWEXTEX
EMR_SETVIEWPORTEXTEX
EMR_SETPOLYFILLMODE
EMR_CREATEBRUSHINDIRECT
```

Draw: [✓ Draw All] Single-Step To Selection

Rectangle:
[✓ Header] 43, -976, 784, -68
Client 0, 0, 228, 471
Header Aspect 741, 908 — 1 — 741, 908
Specific T 0 L 0 R 0

# Toolset – reading & writing (pyemf)

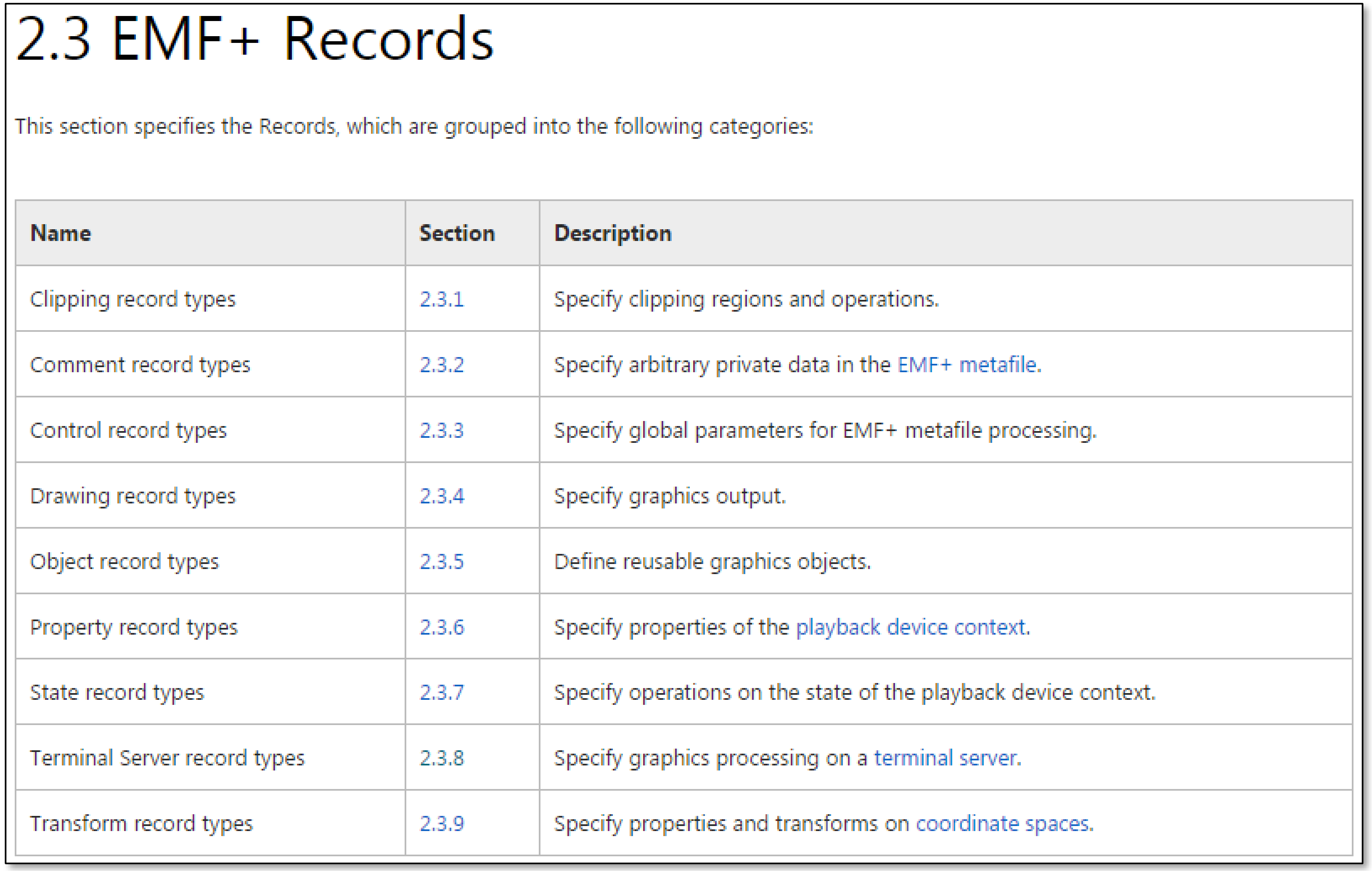
```python
#!/usr/bin/env python
import os
import pyemf
import sys

def main(argv):
  if len(argv) != 2:
    print "Usage: %s /path/to/poc.emf" % argv[0]
    sys.exit(1)

  emf = pyemf.EMF(width = 100, height = 100, density = 1)
  emf.CreateSolidBrush(0x00ff00)
  emf.SelectObject(1)
  emf.Polygon([(0, 0), (0, 100), (100, 100), (100, 0)])

  emf.save(argv[1])

if __name__ == "__main__":
  main(sys.argv)
```

## The latest: EMF+

- GDI had all the fundamental primitives, but lacked many complex features (anti-aliasing, floating point coords, support for JPEG/PNG etc.).
- Windows XP introduced a more advanced library called GDI+ in 2001.
  - Built as a user-mode gdiplus.dll library, mostly on top of regular GDI (gdi32.dll).
  - Provides high-level interfaces for C++ and .NET, therefore is much easier to use.
  - GDI+ itself is written in C++, so all the typical memory corruption bugs still apply.

## The latest: EMF+

- Since there is a new interface, there must also be a new image format with its serialized calls.
- Say hi to EMF+!
- Basically same as EMF, but representing GDI+ calls.
- Come in two flavours: EMF+ Only and EMF+ Dual.
  - „Only” contains exclusively GDI+ records, and can only be displayed with GDI+.
  - „Dual” stores the picture with two sets of records, compatible with both GDI/GDI+ clients.



## 2.3 EMF+ Records

This section specifies the Records, which are grouped into the following categories:

| Name | Section | Description |
|---|---|---|
| Clipping record types | 2.3.1 | Specify clipping regions and operations. |
| Comment record types | 2.3.2 | Specify arbitrary private data in the EMF+ metafile. |
| Control record types | 2.3.3 | Specify global parameters for EMF+ metafile processing. |
| Drawing record types | 2.3.4 | Specify graphics output. |
| Object record types | 2.3.5 | Define reusable graphics objects. |
| Property record types | 2.3.6 | Specify properties of the playback device context. |
| State record types | 2.3.7 | Specify operations on the state of the playback device context. |
| Terminal Server record types | 2.3.8 | Specify graphics processing on a terminal server. |
| Transform record types | 2.3.9 | Specify properties and transforms on coordinate spaces. |

# Formats and implementations in Windows

- Three formats in total to consider: WMF, EMF, EMF+.
- Three libraries: GDI, GDI+ and `MF3216`.
  - MF3216.DLL is a system library with just one meaningful exported function: `ConvertEmfToWmf`.
  - Used for the automatic conversion between WMF/EMF formats in the Windows clipboard.
    - „Synthesized” formats `CF_METAFILEPICT` and `CF_ENHMETAFILE`.
  - No bugs found there. ☹

## Formats and implementations in Windows

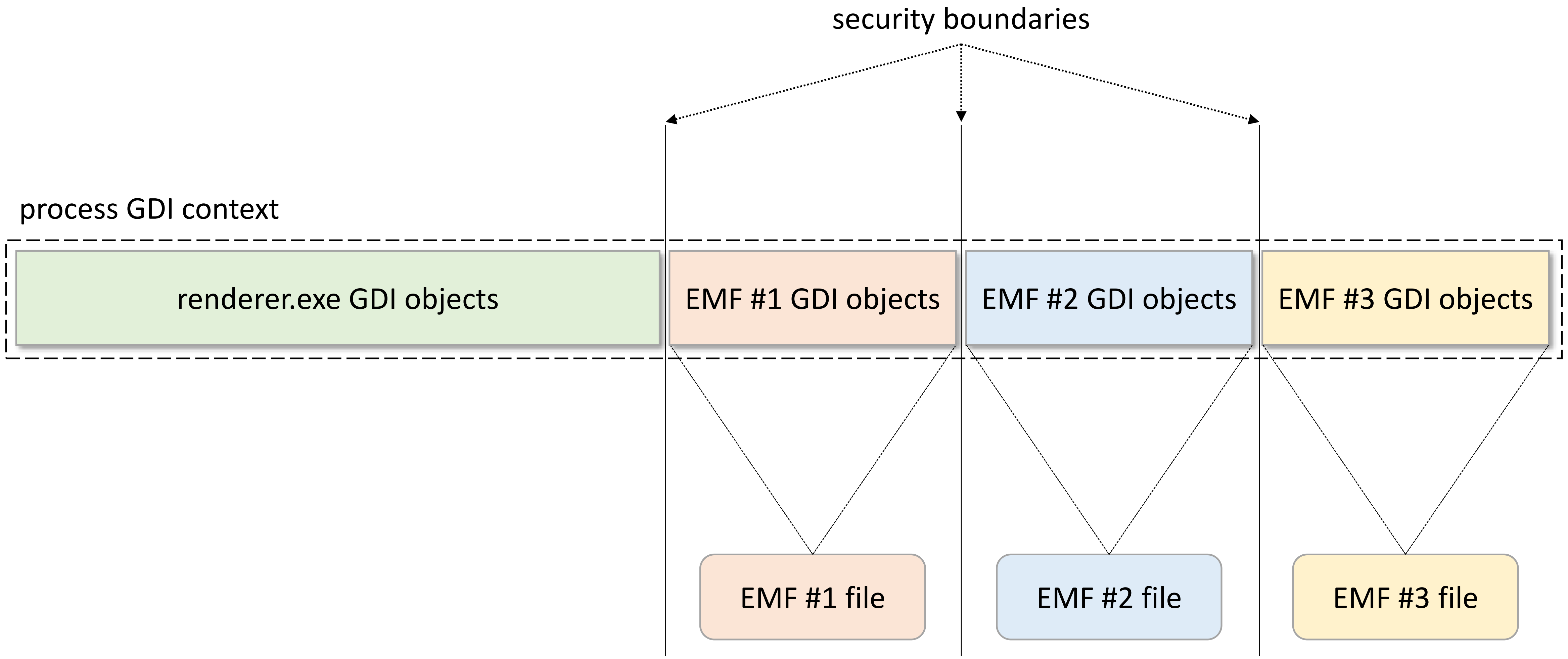
| Library | Supported formats |
|---|---|
| GDI | WMF, EMF |
| GDI+ | WMF, EMF, EMF+ |
| MF3216 | EMF |

In this talk, we'll focus on auditing and exploiting the EMF parts, as this is where the most (interesting) issues were discovered.

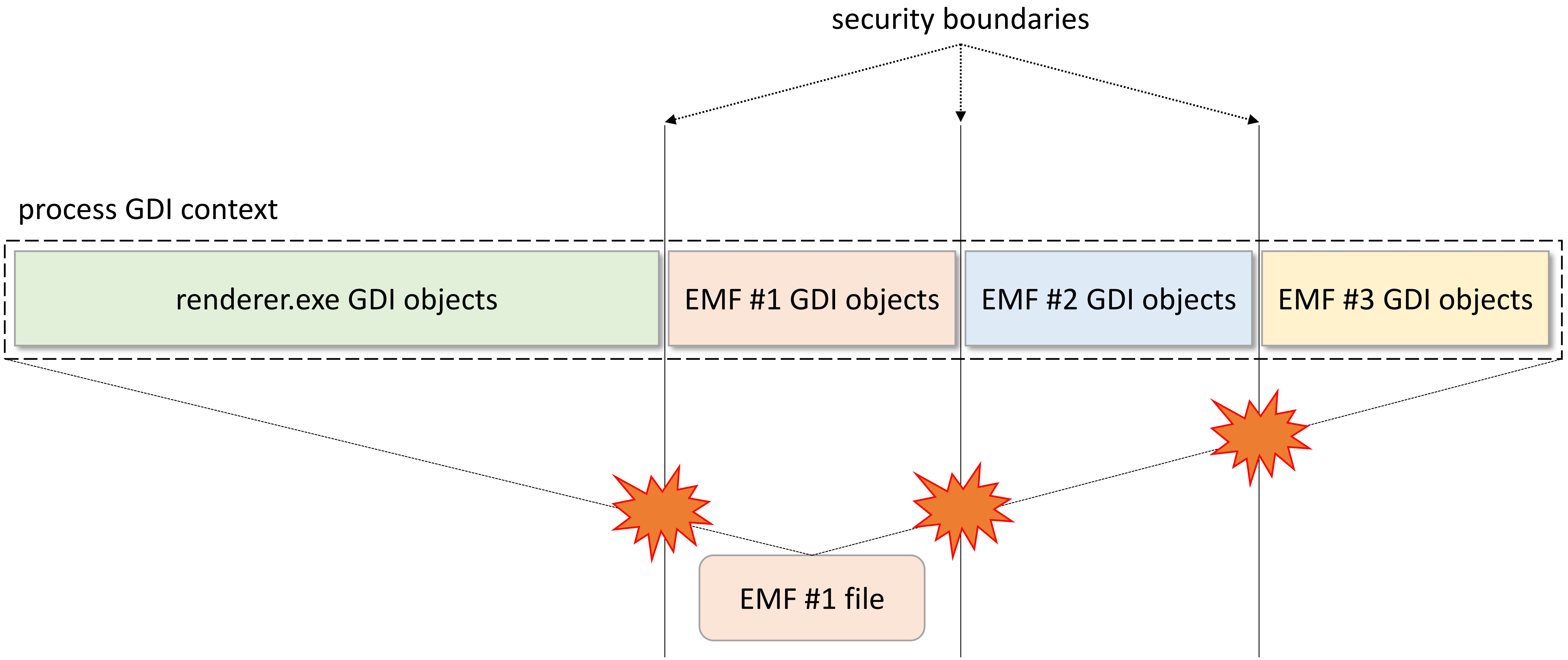
## Attack scenario

- In all cases, Metafiles are processed in the user-mode context of the renderer process, in the corresponding DLL.
    - GDI, GDI+ and MF3216 iterate through all input records and translate them into GDI/GDI+ calls.
- Memory corruption bugs will result in arbitrary code execution in that context.
- **Important:** Metafiles directly operate on the GDI context of the renderer.
    - Can create, delete, change and use various GDI objects on behalf of the process.
    - In theory, it should only have access to its own objects and be self-contained.
    - However, any bugs in the implementation could enable access to *external* graphics objects used by the program.
    - A peculiar case of „privilege escalation”.

## Attack scenario: GDI context priv. escal.



# Attack scenario: GDI context priv. escal.



## Types of Metafile bugs

1. **Memory corruption bugs**
   - Buffer overflows etc. due to mishandling specific records.
   - Potentially exploitable in any type of renderer.
   - Impact: typically RCE.
2. **Memory disclosure bugs**
   - Rendering uninitialized or out-of-bounds heap memory as image pixels.
   - Exploitable only in contexts where displayed images can be read back (web browsers, remote renderers).
   - Impact: information disclosure (stealing secret information, defeating ASLR etc.).
3. **Invalid interaction with the OS and GDI object mismanagement.**
   - Impact, exploitability = ???, depending on the specific nature of the bug.

## Let’s get started!

- Earlier this year, I started manually auditing the available EMF implementations.
- This has resulted in 10 CVEs from Microsoft and 3 CVEs from VMware (covering several dozen of actual bugs).
- Let’s look into the root causes and exploitation of the most interesting ones.
  - Examples are shown based on Windows 7 32-bit, but most of the research applies to both bitnesses and versions up to Windows 10.

## Auditing GDI

## Getting started

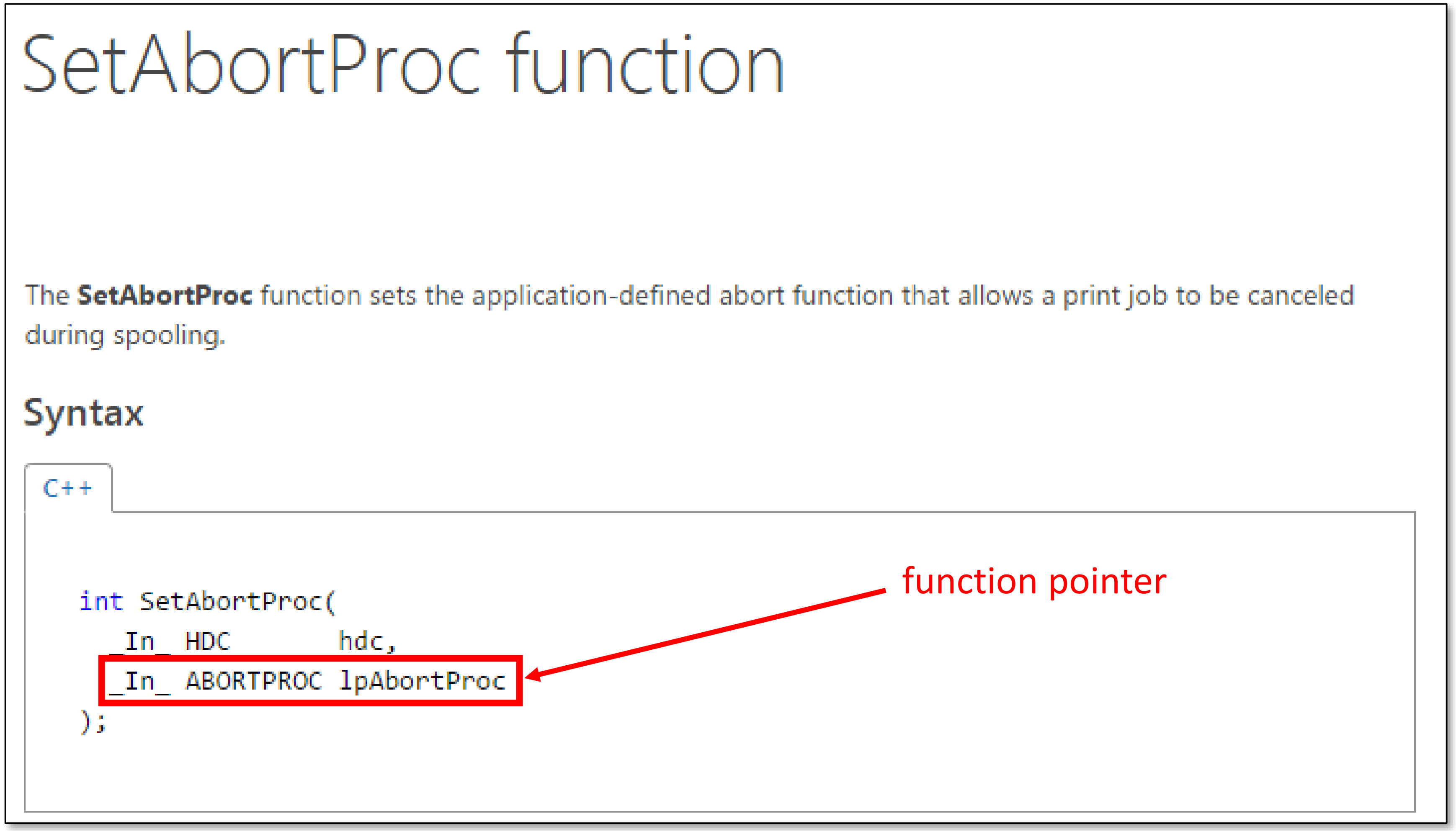
- To get some general idea of where the functionality in question is implemented and what types of bugs were found in the past, it makes sense to check prior art.
- A „wmf vulnerability” query yields just one result:

**the SetAbortProc bug!**

## SetAbortProc WMF bug (CVE-2005-4560)

- Discovered on December 27, 2005. Fixed on January 5, 2006.
- Critical bug, allowed 100% reliable RCE while using GDI to display the exploit (e.g. in Internet Explorer).
- Called „Windows Metafile vulnerability”, won Pwnie Award 2007.
- No memory corruption involved, only documented features of WMF.
- So what was the bug?

# The GDI API...



## SetAbortProc function

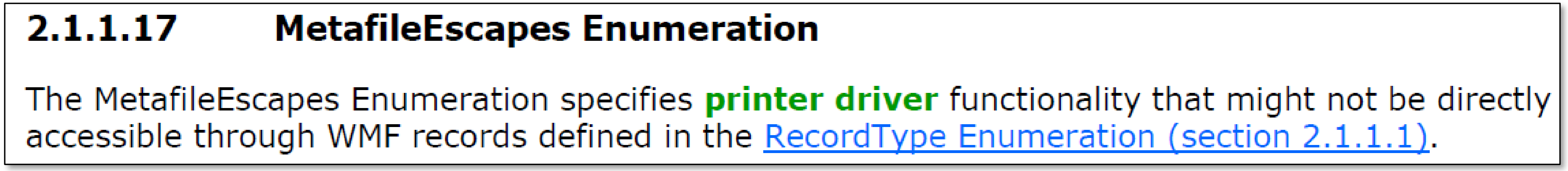
The **SetAbortProc** function sets the application-defined abort function that allows a print job to be canceled during spooling.

### Syntax

C++

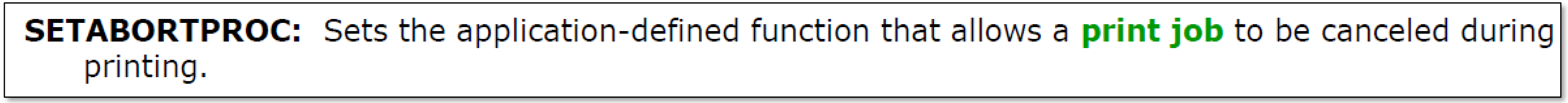
```
int SetAbortProc(
  _In_ HDC       hdc,
  _In_ ABORTPROC lpAbortProc
);
```

# … and the WMF counterpart



**2.1.1.17 MetafileEscapes Enumeration**

The MetafileEscapes Enumeration specifies **printer driver** functionality that might not be directly accessible through WMF records defined in the RecordType Enumeration (section 2.1.1.1).



**SETABORTPROC:** Sets the application-defined function that allows a **print job** to be canceled during printing.

## In essence...

... the format itself supported calling:

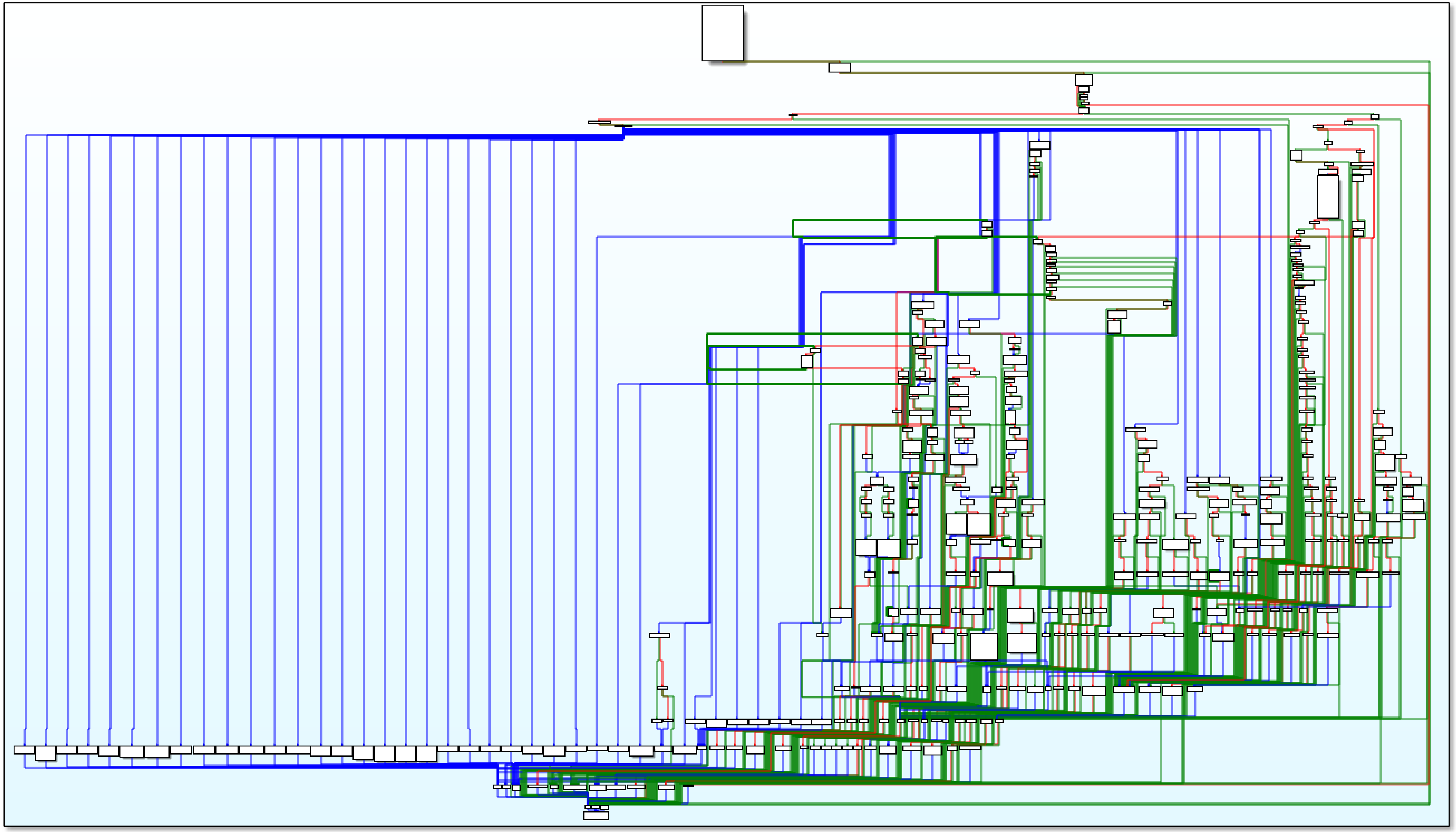
```
SetAbortProc(hdc, (ABORTPROC)"controlled data");
```

and having the function pointer called afterwards.

Code execution by design.

## Lessons learned

1. The format may (un)officially *proxy* calls to interesting / dangerous API calls, so the semantics of each function and its parameters should be checked for unsafe behavior.
2. The handling of WMF takes place in a giant switch/case in `gdi32!PlayMetaFileRecord`.



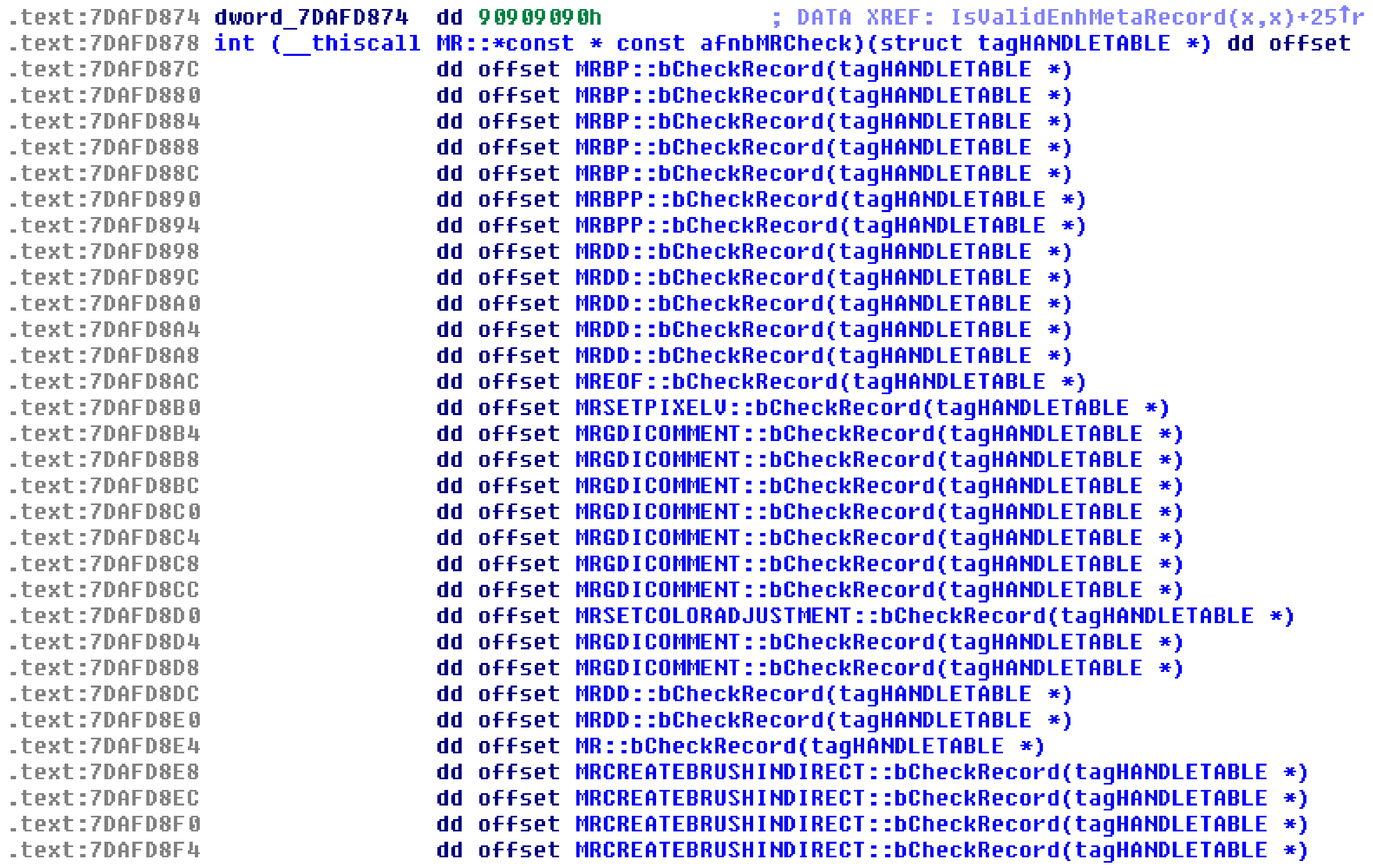
## What about EMF bugs?

- Searching for „emf vulnerability” yields more diverse results.
- Most recent one: „Yet Another Windows GDI Story” by Hossein Lotfi (@hosselot).
  - Fixed in April 2015 as part of MS15-035, assigned CVE-2015-1645.
  - A heap-based buffer overflow due to an unchecked assumption about an input „size” field in one of the records (SETDIBITSTODEVICE).
  - In large part an inspiration to start looking into EMF security myself.

## Lessons learned

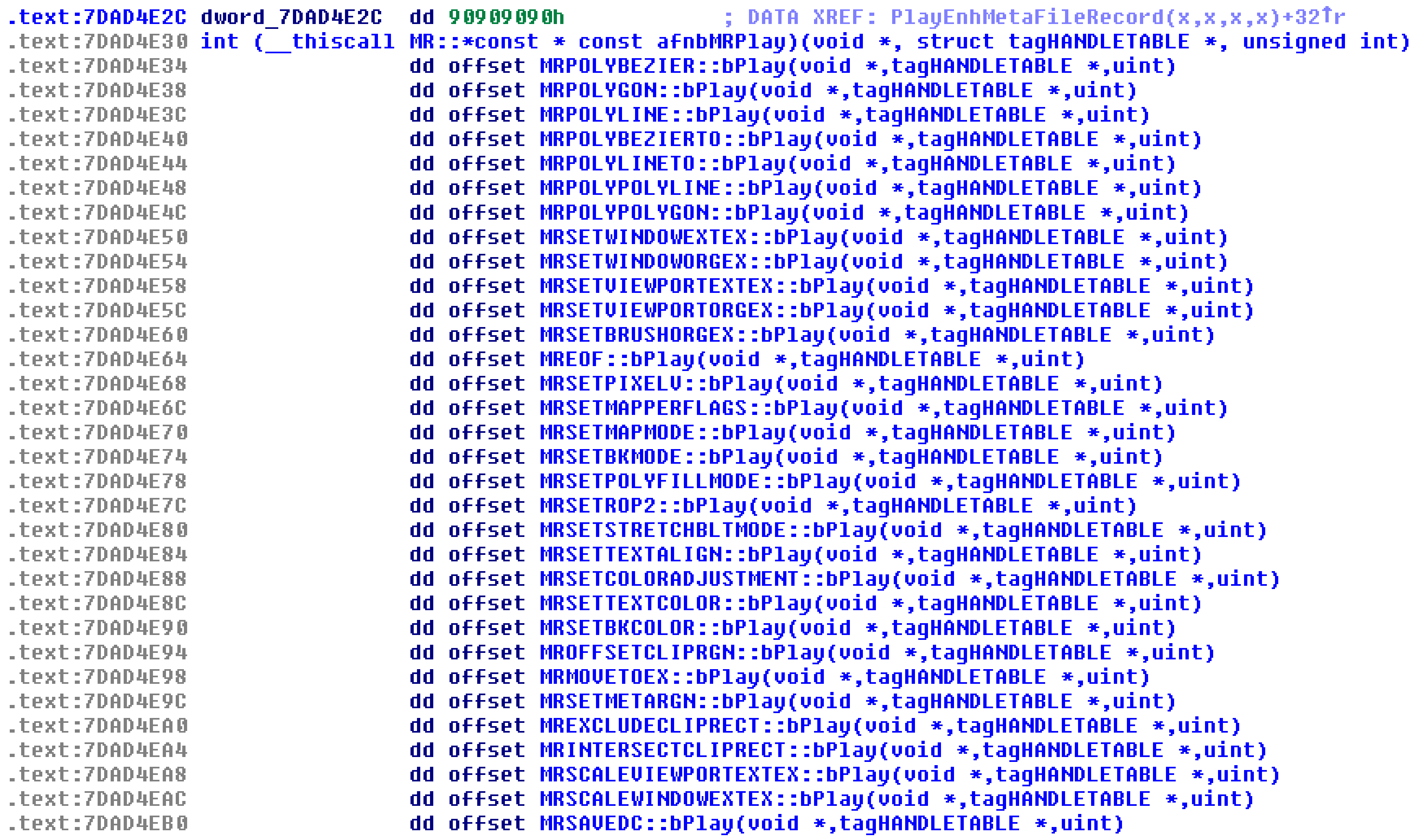
- Main function for playing EMF records is `gdi32!PlayEnhMetaFileRecord`.
- Each record type has its own class with two methods:
  - `::bCheckRecord()` – checks the internal integrity and correctness of the record.
  - `::bPlay()` – performs the actions indicated in the record.

# GDI32 ::bCheckRecord array



```
.text:7DAFD874 dword_7DAFD874  dd 90909090h             ; DATA XREF: IsValidEnhMetaRecord(x,x)+25↑r
.text:7DAFD878 int (__thiscall MR::*const * const afnbMRCheck)(struct tagHANDLETABLE *) dd offset
.text:7DAFD87C                 dd offset MRBP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD880                 dd offset MRBP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD884                 dd offset MRBP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD888                 dd offset MRBP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD88C                 dd offset MRBP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD890                 dd offset MRBPP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD894                 dd offset MRBPP::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD898                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD89C                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8A0                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8A4                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8A8                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8AC                 dd offset MREOF::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8B0                 dd offset MRSETPIXELV::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8B4                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8B8                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8BC                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8C0                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8C4                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8C8                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8CC                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8D0                 dd offset MRSETCOLORADJUSTMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8D4                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8D8                 dd offset MRGDICOMMENT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8DC                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8E0                 dd offset MRDD::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8E4                 dd offset MR::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8E8                 dd offset MRCREATEBRUSHINDIRECT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8EC                 dd offset MRCREATEBRUSHINDIRECT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8F0                 dd offset MRCREATEBRUSHINDIRECT::bCheckRecord(tagHANDLETABLE *)
.text:7DAFD8F4                 dd offset MRCREATEBRUSHINDIRECT::bCheckRecord(tagHANDLETABLE *)
```

# GDI32 ::bPlay array



```
.text:7DAD4E2C dword_7DAD4E2C  dd 90909090h            ; DATA XREF: PlayEnhMetaFileRecord(x,x,x,x)+32↑r
.text:7DAD4E30 int (__thiscall MR::*const * const afnbMRPlay)(void *, struct tagHANDLETABLE *, unsigned int)
.text:7DAD4E34                 dd offset MRPOLYBEZIER::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E38                 dd offset MRPOLYGON::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E3C                 dd offset MRPOLYLINE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E40                 dd offset MRPOLYBEZIERTO::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E44                 dd offset MRPOLYLINETO::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E48                 dd offset MRPOLYPOLYLINE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E4C                 dd offset MRPOLYPOLYGON::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E50                 dd offset MRSETWINDOWEXTEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E54                 dd offset MRSETWINDOWORGEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E58                 dd offset MRSETVIEWPORTEXTEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E5C                 dd offset MRSETVIEWPORTORGEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E60                 dd offset MRSETBRUSHORGEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E64                 dd offset MREOF::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E68                 dd offset MRSETPIXELV::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E6C                 dd offset MRSETMAPPERFLAGS::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E70                 dd offset MRSETMAPMODE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E74                 dd offset MRSETBKMODE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E78                 dd offset MRSETPOLYFILLMODE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E7C                 dd offset MRSETROP2::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E80                 dd offset MRSETSTRETCHBLTMODE::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E84                 dd offset MRSETTEXTALIGN::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E88                 dd offset MRSETCOLORADJUSTMENT::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E8C                 dd offset MRSETTEXTCOLOR::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E90                 dd offset MRSETBKCOLOR::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E94                 dd offset MROFFSETCLIPRGN::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E98                 dd offset MRMOVETOEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4E9C                 dd offset MRSETMETARGN::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4EA0                 dd offset MREXCLUDECLIPRECT::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4EA4                 dd offset MRINTERSECTCLIPRECT::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4EA8                 dd offset MRSCALEVIEWPORTEXTEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4EAC                 dd offset MRSCALEWINDOWEXTEX::bPlay(void *,tagHANDLETABLE *,uint)
.text:7DAD4EB0                 dd offset MRSAVEDC::bPlay(void *,tagHANDLETABLE *,uint)
```

That’s a starting point.

## CVE-2016-0168

| | |
|---|---|
| **Impact:** | File Existence Information Disclosure |
| **Record:** | EMR_CREATECOLORSPACE, EMR_CREATECOLORSPACEW |
| **Exploitable in:** | Internet Explorer |
| **CVE:** | CVE-2016-0168 |
| ***google-security-research* entry:** | 722 |
| **Fixed:** | MS16-055, 10 May 2016 |

# Minor bug #1 in EMR_CREATECOLORSPACEW

- The quality of the code can be immediately recognized by observing many small, but obvious bugs.
- `MRCREATECOLORSPACEW::bCheckRecord()` checks that the size of the record is ≥ 0x50 bytes long:

```
.text:7DB01AEF     mov     eax, [esi+4]
.text:7DB01AF2     cmp     eax, 50h
.text:7DB01AF5     jb      short loc_7DB01B1E
```

- Then immediately proceeds to read a .cbData field at offset 0x25C:

```
.text:7DB01AF7     mov     ecx, [esi+25Ch]
```

- Result: out-of-bounds read by 0x20C bytes.

# Minor bug #2 in EMR_CREATECOLORSPACEW

- Then, the `.cbData` from invalid offset 0x25C is used to verify the record length:

```
.text:7DB01AF7    mov     ecx, [esi+25Ch]
.text:7DB01AFD    add     ecx, 263h
.text:7DB01B03    and     ecx, 0FFFFFFFCh
.text:7DB01B06    cmp     eax, ecx
.text:7DB01B08    ja      short loc_7DB01B1E
```

- The above translates to:

```
if (... && record.length <= ((record->cbData + 0x263) & ~3) && ...) {
  // Record valid.
}
```

# Minor bug #2 in EMR_CREATECOLORSPACEW

- Two issues here:
  1. Obvious integer overflow making a large `.cbData` pass the check.
  2. Why would the record length be **smaller** then the data declared within? It should be **larger**!
- It all doesn’t matter anyway, since the data is not used in any further processing.

## Minor bug #3 in EMR_CREATECOLORSPACEW

- The `.lcsFilename` buffer of the user-defined `LOGCOLORSPACEW` structure is not verified to be nul-terminated.
  - May lead to out-of-bound reads while accessing the string.
- As clearly visible, there are lots of unchecked assumptions in the implementation, even though only minor so far.
  - Keeps our hopes up for something more severe.

## The file existence disclosure

- Back to the functionality of EMR_CREATECOLORSPACE[W] records: all they do is call `CreateColorSpace[W]` with a fully controlled `LOGCOLORSPACE` structure:

```
typedef struct tagLOGCOLORSPACE {
  DWORD         lcsSignature;
  DWORD         lcsVersion;
  DWORD         lcsSize;
  LCSCSTYPE     lcsCSType;
  LCSGAMUTMATCH lcsIntent;
  CIEXYZTRIPLE  lcsEndpoints;
  DWORD         lcsGammaRed;
  DWORD         lcsGammaGreen;
  DWORD         lcsGammaBlue;
  TCHAR         lcsFilename[MAX_PATH];
} LOGCOLORSPACE, *LPLOGCOLORSPACE;
```

# Inside CreateColorSpaceW

- The function builds a color profile file path using internal `gdi32!BuildIcmProfilePath`.
  - if the provided filename is relative, it is appended to a system directory path.
  - otherwise, absolute paths are left as-is.
- All paths are accepted, except for those starting with two "/" or "\" characters:

```
if ((pszSrc[0] == '\\' || pszSrc[0] == '/') &&
    (pszSrc[1] == '\\' || pszSrc[1] == '/')) {
  // Path denied.
}
```

## Inside CreateColorSpaceW

- This is supposedly to prevent specifying remote UNC paths starting with the "\\" prefix, e.g. \\192.168.1.13\C\Users\test\profile.icc.
- However, James Forshaw noted that this check is not effective, as the prefix can be also represented as "\??\UNC\".
- The check is easily bypassable with:

\??\UNC\192.168.1.13\C\Users\test\profile.icc

## CreateColorSpaceInternalW: last step

- After the path is formed, but before invoking the `NtGdiCreateColorSpace` system call, the function opens the file and immediately closes it to see if it exists:

```
HANDLE hFile = CreateFileW(&FileName, GENERIC_READ, FILE_SHARE_READ, 0,
                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
if (hFile == INVALID_HANDLE_VALUE) {
  GdiSetLastError(2016);
  return 0;
}
CloseHandle(hFile);
```

## Consequences

- In result, we can have `CreateFileW()` called over any chosen path.
  - If it succeeds, the color space object is created and the function returns success.
  - If it fails, the GDI object is not created and the handler returns failure.
- Sounds like information disclosure potential.
  - How do we approach exploitation e.g. in Internet Explorer?

## Intuitive way: leaking the return value

- Since the return value of `CreateFileW()` determines the success of the record processing, we could maybe leak this bit?
  - Initial idea: use `EMR_CREATECOLORSPACE` as the first record, followed by a drawing operation.
  - If the drawing is never executed (which can be determined with the <canvas> tag), the call failed.

## Intuitive way: leaking the return value

- Unfortunately impossible.
- The `gdi32!_bInternalPlayEMF` function (called by `PlayEnhMetaFile` itself) doesn’t abort image processing when one record fails.
  - A „success” flag is set to FALSE, and the function proceeds to further operations.
- All records are always executed, and the return value is a flag indicating if at least one of the records failed during the process.

## Can’t we leak the final return value?

- No, not really.
- The return value of `PlayEnhMetaFile` is discarded by Internet Explorer in `mshtml!CImgTaskEmf::Decode`:

```
.text:64162B49        call     ds:__imp__PlayEnhMetaFile@12
.text:64162B4F        or       dword ptr [ebx+7Ch], 0FFFFFFFFh
.text:64162B53        lea      eax, [esp+4C8h+var_49C]
```

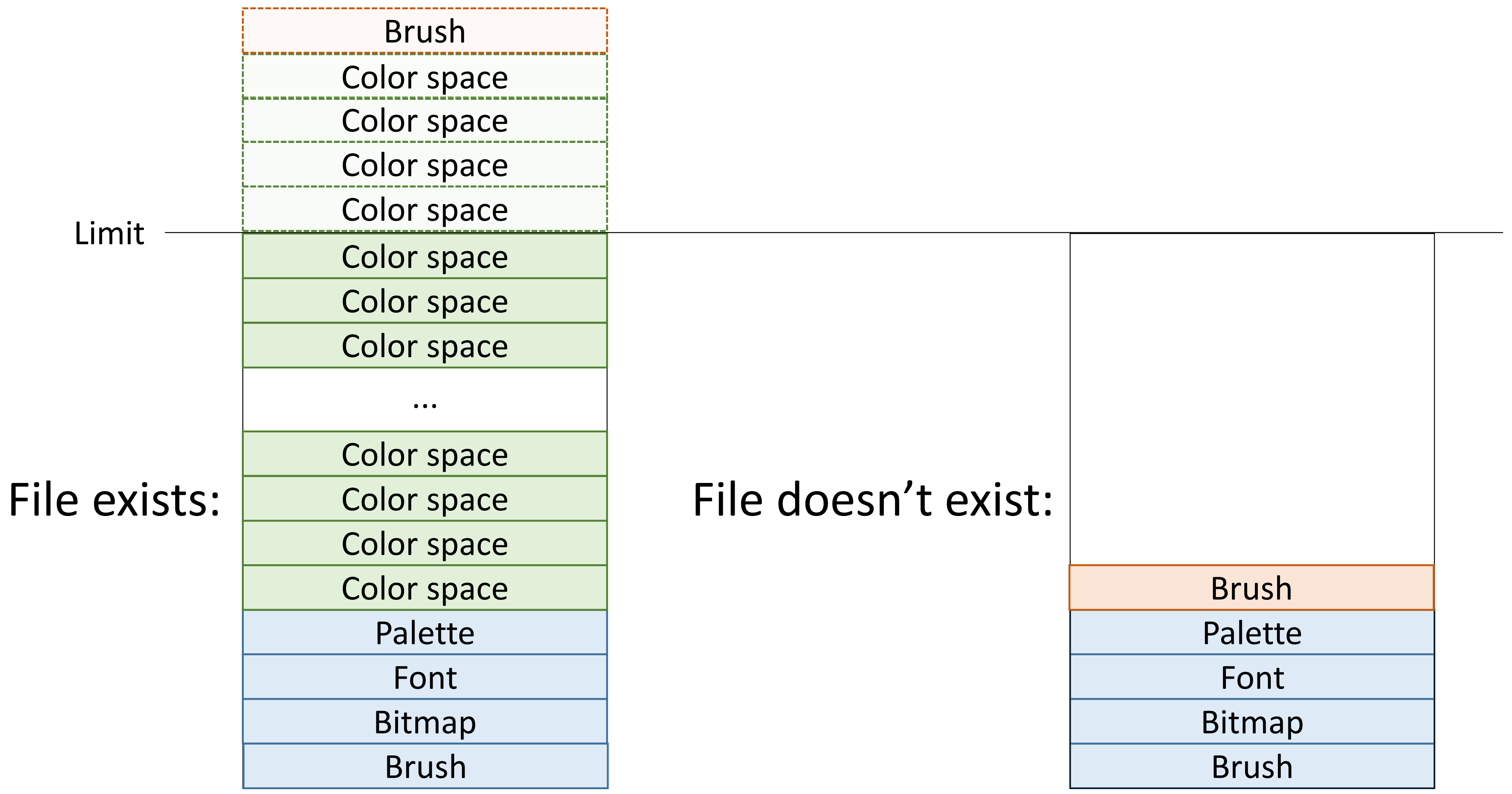
## Other disclosure options

- The other indicator could be the creation of a color space object via `NtGdiCreateColorSpace`.
- Leaking it directly is not easy (if at all possible), but maybe there is some side channel?

## Using the GDI object limit

- Every process in Windows is limited to max. 10,000 GDI objects by default.
  - The number can be adjusted in the registry, but isn’t for IE.
- If we use 10,000 `EMR_CREATECOLORSPACEW` records with the file path we want to check, then:
  - If the file exists, we’ll have 10,000 color space objects, reaching the per-process limit.
  - If it doesn’t, we won’t have any color spaces at all.
- We’re now either at the limit, or not. If we then create a brush (one more object) and try to paint, then:
  - If the file exists, the brush creation will fail and the default brush will be used.
  - If it doesn’t, the brush will be created and used for paiting.

# GDI object limit as oracle illustrated



# DEMO

## Vulnerability impact

- Arbitrary file existence disclosure, useful for many purposes:
    - Recognizing specific software (and versions) that the user has installed, for targetted attacks.
    - Tracking users (by creating profiles based on existing files).
    - Tracking the opening times of offline documents (e.g. each opening in Microsoft Office could trigger a ping to remote server via SMB).
    - Blindly scanning network shares available to the user.

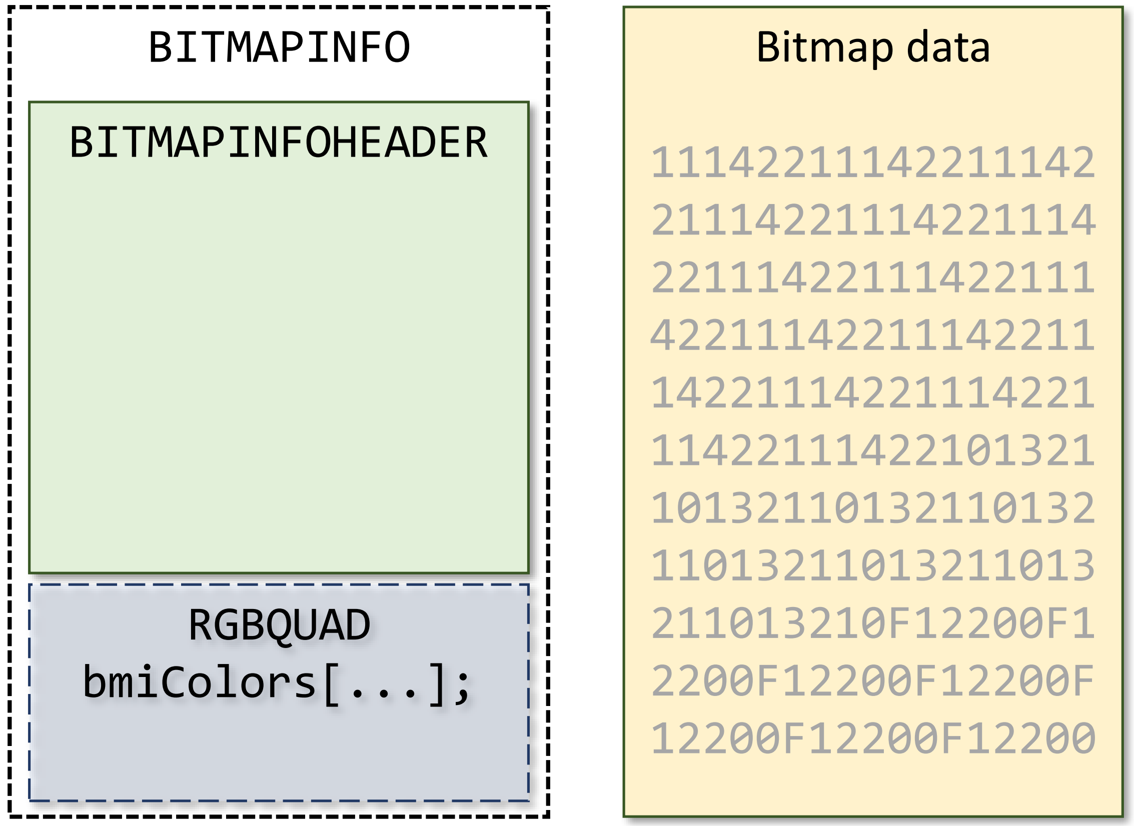
## CVE-2016-3216

| | |
|---|---|
| **Impact:** | Memory disclosure |
| **Record:** | Multiple records (10) |
| **Exploitable in:** | Internet Explorer |
| **CVE:** | CVE-2016-3216 |
| ***google-security-research* entry:** | 757 |
| **Fixed:** | MS16-074, 14 June 2016 |

# Device Independent Bitmaps (DIBs)

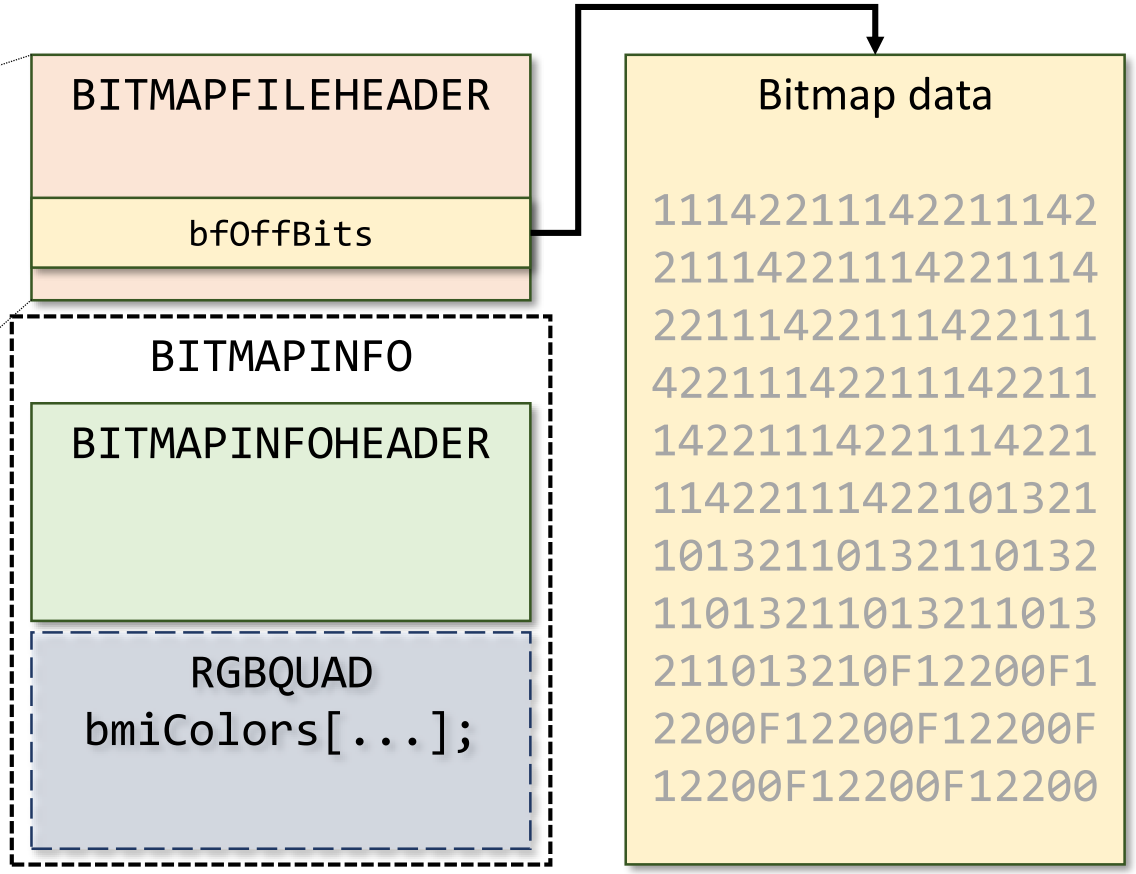
In Windows GDI, raster bitmaps are usually stored in memory in the form of DIBs:

- Short header containing basic metadata about the image, followed by optional palette.
- The image data itself.



## .BMP files are just DIBs, too.

```
typedef struct tagBITMAPFILEHEADER {
  WORD  bfType;
  DWORD bfSize;
  WORD  bfReserved1;
  WORD  bfReserved2;
  DWORD bfOffBits;
} BITMAPFILEHEADER;
```



## BITMAPINFOHEADER, the trivial header

```
typedef struct tagBITMAPINFOHEADER {
  DWORD biSize;
  LONG  biWidth;
  LONG  biHeight;
  WORD  biPlanes;
  WORD  biBitCount;
  DWORD biCompression;
  DWORD biSizeImage;
  LONG  biXPelsPerMeter;
  LONG  biYPelsPerMeter;
  DWORD biClrUsed;
  DWORD biClrImportant;
} BITMAPINFOHEADER;
```

- Short and simple structure.
- 40 bytes in length (in typical form).
- Only 8 meaningful fields.

## Is it really so trivial to handle?

- `biSize` needs to be sanitized (can only be a few valid values).
- `biWidth`, `biHeight`, `biPlanes`, `biBitCount` can cause integer overflows (often multiplied with each other).
- `biHeight` can be negative to indicate bottom-up bitmap.
- `biPlanes` must be 1.
- `biBitCount` must be one of {1, 2, 4, 8, 16, 24, 32}.
  - For `biBitCount` < 16, a color palette can be used.
  - The size of the color palette is also influenced by `biClrUsed`.

## Is it really so trivial to handle?

- `biCompression` can be `BI_RGB`, `BI_RLE8`, `BI_RLE4`, `BI_BITFIELDS`, ...
  - Each compression scheme must be handled correctly.
- `biSizeImage` must correspond to the actual image size.
- The palette must be sufficiently large to contain all entries.
- The pixel data buffer must be sufficiently large to describe all pixels.
- Encoded pixels must correspond to the values in header (e.g. not exceed the palette size etc.).

## Many potential problems

1. The decision tree for correctly handling a DIB based on its header is very complex.
2. Lots of corner cases to cover and implementation bugs to avoid.
3. A consistent handling across various parts of code is required.

# GDI functions operating on DIB (directly)

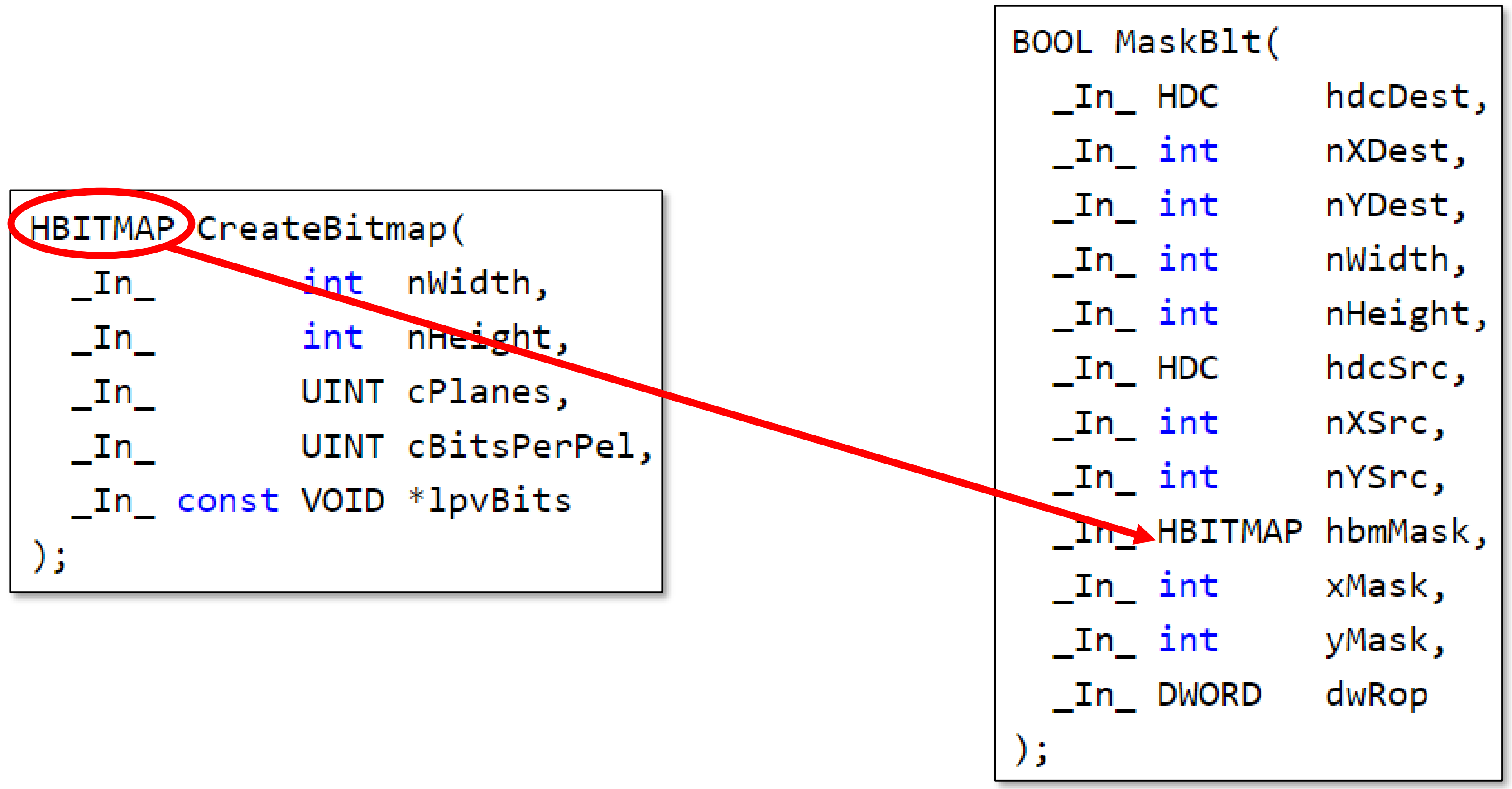
```
int StretchDIBits(
  _In_          HDC         hdc,
  _In_          int         XDest,
  _In_          int         YDest,
  _In_          int         nDestWidth,
  _In_          int         nDestHeight,
  _In_          int         XSrc,
  _In_          int         YSrc,
  _In_          int         nSrcWidth,
  _In_          int         nSrcHeight,
  _In_   const  VOID        *lpBits,
  _In_   const  BITMAPINFO  *lpBitsInfo,
  _In_          UINT        iUsage,
  _In_          DWORD       dwRop
);
```

pointer to image data

pointer to DIB header

```
int SetDIBitsToDevice(
  _In_          HDC         hdc,
  _In_          int         XDest,
  _In_          int         YDest,
  _In_          DWORD       dwWidth,
  _In_          DWORD       dwHeight,
  _In_          int         XSrc,
  _In_          int         YSrc,
  _In_          UINT        uStartScan,
  _In_          UINT        cScanLines,
  _In_   const  VOID        *lpvBits,
  _In_   const  BITMAPINFO  *lpbmi,
  _In_          UINT        fuColorUse
);
```

# GDI functions operating on DIB (indirectly)



```
HBITMAP CreateBitmap(
   _In_       int   nWidth,
   _In_       int   nHeight,
   _In_       UINT  cPlanes,
   _In_       UINT  cBitsPerPel,
   _In_ const VOID *lpvBits
);
```

```
BOOL MaskBlt(
   _In_ HDC     hdcDest,
   _In_ int     nXDest,
   _In_ int     nYDest,
   _In_ int     nWidth,
   _In_ int     nHeight,
   _In_ HDC     hdcSrc,
   _In_ int     nXSrc,
   _In_ int     nYSrc,
   _In_ HBITMAP hbmMask,
   _In_ int     xMask,
   _In_ int     yMask,
   _In_ DWORD   dwRop
);
```

## Data sanitization responsibility

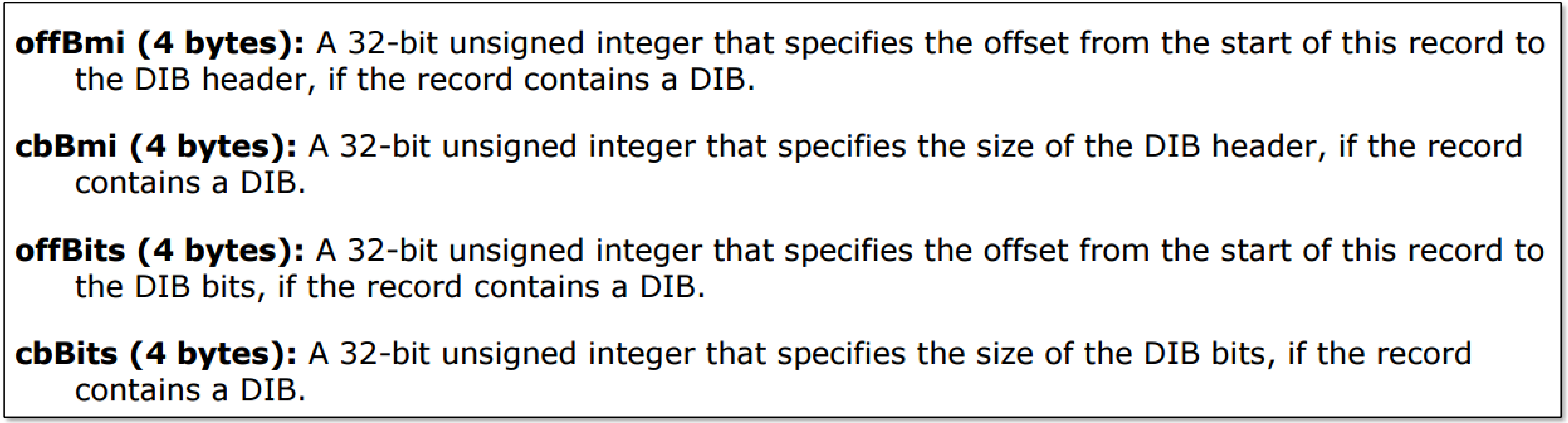
- In all cases, it is the API caller’s resposibility to make sure the headers and data are correct and adequate.
- Passing in fully user-controlled input data is somewhat problematic, as the application code would have to „clone” GDI’s DIB handling.
- Guess what? EMF supports multiple records which contain embedded DIBs.

## EMF records containing DIBs

- EMR_ALPHABLEND
- EMR_BITBLT
- EMR_MASKBLT
- EMR_PLGBLT
- EMR_STRETCHBLT
- EMR_TRANSPARENTBLT
- EMR_SETDIBITSTODEVICE
- EMR_STRETCHDIBITS
- EMR_CREATEMONOBRUSH
- EMR_EXTCREATEPEN

## The common scheme

- Two pairs of `(offset, size)` for both the header and the bitmap:



**offBmi (4 bytes):** A 32-bit unsigned integer that specifies the offset from the start of this record to the DIB header, if the record contains a DIB.

**cbBmi (4 bytes):** A 32-bit unsigned integer that specifies the size of the DIB header, if the record contains a DIB.

**offBits (4 bytes):** A 32-bit unsigned integer that specifies the offset from the start of this record to the DIB bits, if the record contains a DIB.

**cbBits (4 bytes):** A 32-bit unsigned integer that specifies the size of the DIB bits, if the record contains a DIB.

# Necessary checks in the EMF record handlers

- In each handler dealing with DIBs, there are four necessary consistency checks:
    1. `cbBmiSrc` is adequately large for the header to fit in.
    2. `(offBmiSrc, offBmiSrc + cbBmiSrc)` resides fully within the record.
    3. `cbBitsSrc` is adequately large for the bitmap data to fit in.
    4. `(offBitsSrc, offBitsSrc + cbBitsSrc)` resides fully within the record.

# Checks were missing in many combinations

| Record handlers | Missing checks |
|---|---|
| `MRALPHABLEND::bPlay`<br>`MRBITBLT::bPlay`<br>`MRMASKBLT::bPlay`<br>`MRPLGBLT::bPlay`<br>`MRSTRETCHBLT::bPlay`<br>`MRTRANSPARENTBLT::bPlay` | #1, #2 |
| `MRSETDIBITSTODEVICE::bPlay` | #3 |
| `MRSTRETCHDIBITS::bPlay` | #1, #3 |
| `MRSTRETCHDIBITS::bPlay`<br>`MRCREATEMONOBRUSH::bPlay`<br>`MREXTCREATEPEN::bPlay` | #1, #2, #3, #4 |

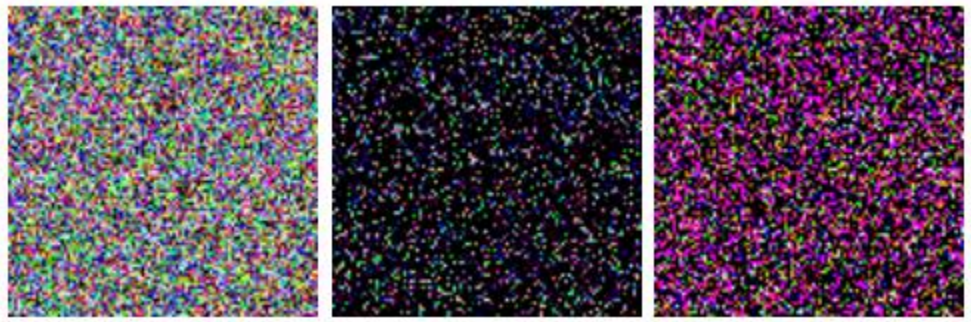
* This was just after a cursory look; Microsoft might have fixed more.

## The consequence

- Due to missing checks, parts of the image description could be loaded from other parts of the process address space (e.g. adjacent heap allocations):
  - DIB header
  - Color palette
  - Pixel data
- Uninitialized or out-of-bound heap memory could be disclosed with the palette or pixel data.

## Proof of concept

- I hacked up a PoC file with an `EMR_STRETCHBLT` record, containing an 8-bpp DIB with palette entries going beyond the file.
- Result: garbage bytes being displayed as image pixels.
- The same picture being displayed three times in a row in IE:



- The data can be read back using HTML5, in order to leak module addresses and other sensitive data.

## DEMO

# Auditing ATMFD.DLL

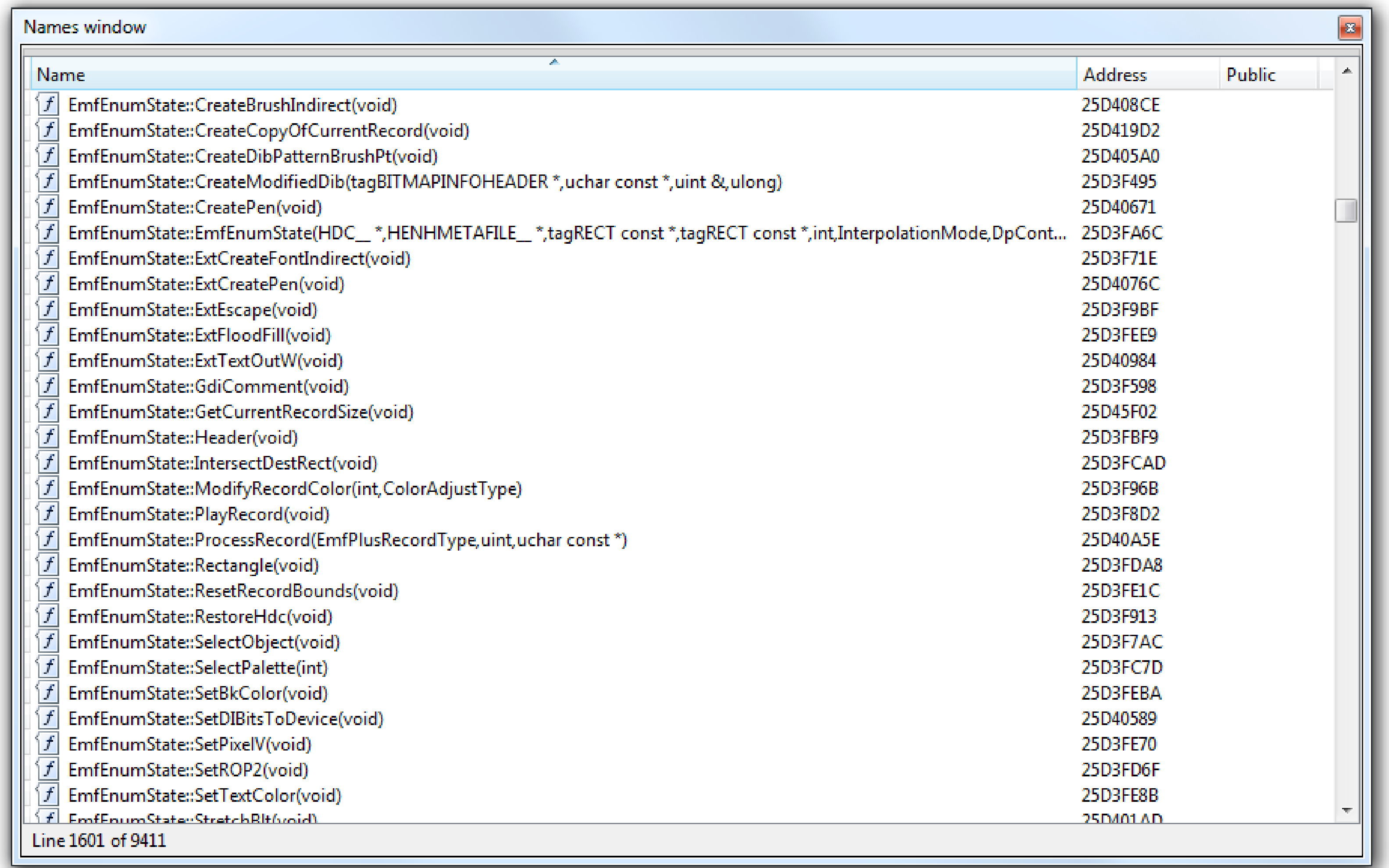
Out of time, please see the full slide deck released after the conference.

## Auditing GDI+

## GDI+ as a viable target

- GDI+ supports both EMF and EMF+.
    - Most of the implementation is independent, but for some parts of the format, it falls back to GDI code.
    - Hence, some GDI bugs could also affect GDI+ clients.
- Most prominent client of GDI+ is the Microsoft Office suite.
- Once again, let’s manually audit the entirety of EMF record handlers.

## Attack surface easy to find

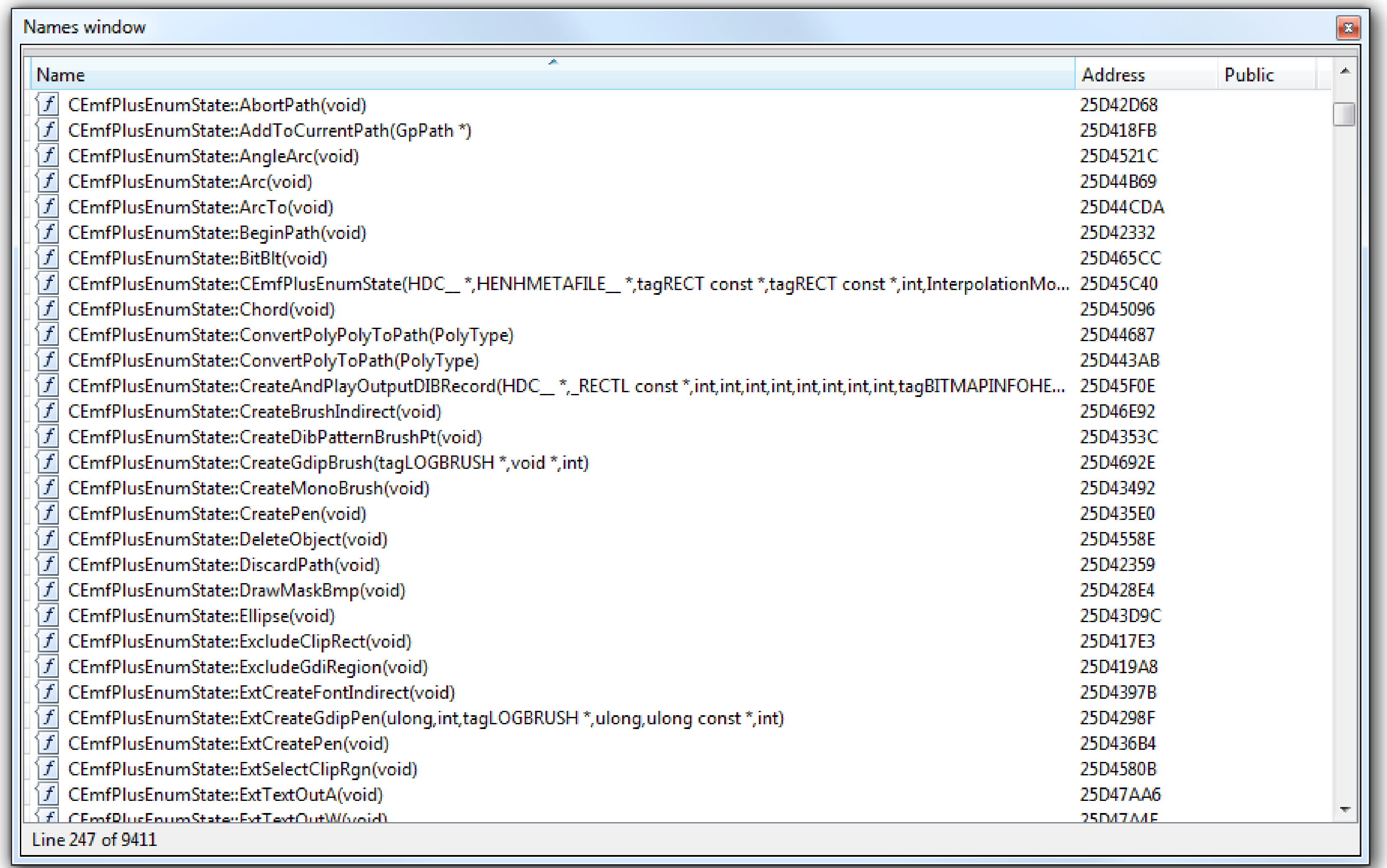


Names window

| Name | Address | Public |
|---|---|---|
| EmfEnumState::CreateBrushIndirect(void) | 25D408CE | |
| EmfEnumState::CreateCopyOfCurrentRecord(void) | 25D419D2 | |
| EmfEnumState::CreateDibPatternBrushPt(void) | 25D405A0 | |
| EmfEnumState::CreateModifiedDib(tagBITMAPINFOHEADER *,uchar const *,uint &,ulong) | 25D3F495 | |
| EmfEnumState::CreatePen(void) | 25D40671 | |
| EmfEnumState::EmfEnumState(HDC__ *,HENHMETAFILE__ *,tagRECT const *,tagRECT const *,int,InterpolationMode,DpCont... | 25D3FA6C | |
| EmfEnumState::ExtCreateFontIndirect(void) | 25D3F71E | |
| EmfEnumState::ExtCreatePen(void) | 25D4076C | |
| EmfEnumState::ExtEscape(void) | 25D3F9BF | |
| EmfEnumState::ExtFloodFill(void) | 25D3FEE9 | |
| EmfEnumState::ExtTextOutW(void) | 25D40984 | |
| EmfEnumState::GdiComment(void) | 25D3F598 | |
| EmfEnumState::GetCurrentRecordSize(void) | 25D45F02 | |
| EmfEnumState::Header(void) | 25D3FBF9 | |
| EmfEnumState::IntersectDestRect(void) | 25D3FCAD | |
| EmfEnumState::ModifyRecordColor(int,ColorAdjustType) | 25D3F96B | |
| EmfEnumState::PlayRecord(void) | 25D3F8D2 | |
| EmfEnumState::ProcessRecord(EmfPlusRecordType,uint,uchar const *) | 25D40A5E | |
| EmfEnumState::Rectangle(void) | 25D3FDA8 | |
| EmfEnumState::ResetRecordBounds(void) | 25D3FE1C | |
| EmfEnumState::RestoreHdc(void) | 25D3F913 | |
| EmfEnumState::SelectObject(void) | 25D3F7AC | |
| EmfEnumState::SelectPalette(int) | 25D3FC7D | |
| EmfEnumState::SetBkColor(void) | 25D3FEBA | |
| EmfEnumState::SetDIBitsToDevice(void) | 25D40589 | |
| EmfEnumState::SetPixelV(void) | 25D3FE70 | |
| EmfEnumState::SetROP2(void) | 25D3FD6F | |
| EmfEnumState::SetTextColor(void) | 25D3FE8B | |
| EmfEnumState::StretchBlt(void) | 25D401AD | |

Line 1601 of 9411

# Attack surface easy to find



Names window

```
Name                                                                                                                        Address    Public
CEmfPlusEnumState::AbortPath(void)                                                                                          25D42D68
CEmfPlusEnumState::AddToCurrentPath(GpPath *)                                                                               25D418FB
CEmfPlusEnumState::AngleArc(void)                                                                                           25D4521C
CEmfPlusEnumState::Arc(void)                                                                                                25D44B69
CEmfPlusEnumState::ArcTo(void)                                                                                              25D44CDA
CEmfPlusEnumState::BeginPath(void)                                                                                          25D42332
CEmfPlusEnumState::BitBlt(void)                                                                                             25D465CC
CEmfPlusEnumState::CEmfPlusEnumState(HDC__ *,HENHMETAFILE__ *,tagRECT const *,tagRECT const *,int,InterpolationMo...        25D45C40
CEmfPlusEnumState::Chord(void)                                                                                              25D45096
CEmfPlusEnumState::ConvertPolyPolyToPath(PolyType)                                                                          25D44687
CEmfPlusEnumState::ConvertPolyToPath(PolyType)                                                                              25D443AB
CEmfPlusEnumState::CreateAndPlayOutputDIBRecord(HDC__ *,_RECTL const *,int,int,int,int,int,int,int,int,tagBITMAPINFOHE...    25D45F0E
CEmfPlusEnumState::CreateBrushIndirect(void)                                                                                25D46E92
CEmfPlusEnumState::CreateDibPatternBrushPt(void)                                                                            25D4353C
CEmfPlusEnumState::CreateGdipBrush(tagLOGBRUSH *,void *,int)                                                                25D4692E
CEmfPlusEnumState::CreateMonoBrush(void)                                                                                    25D43492
CEmfPlusEnumState::CreatePen(void)                                                                                          25D435E0
CEmfPlusEnumState::DeleteObject(void)                                                                                       25D4558E
CEmfPlusEnumState::DiscardPath(void)                                                                                        25D42359
CEmfPlusEnumState::DrawMaskBmp(void)                                                                                        25D428E4
CEmfPlusEnumState::Ellipse(void)                                                                                            25D43D9C
CEmfPlusEnumState::ExcludeClipRect(void)                                                                                    25D417E3
CEmfPlusEnumState::ExcludeGdiRegion(void)                                                                                   25D419A8
CEmfPlusEnumState::ExtCreateFontIndirect(void)                                                                              25D4397B
CEmfPlusEnumState::ExtCreateGdipPen(ulong,int,tagLOGBRUSH *,ulong,ulong const *,int)                                        25D4298F
CEmfPlusEnumState::ExtCreatePen(void)                                                                                       25D436B4
CEmfPlusEnumState::ExtSelectClipRgn(void)                                                                                   25D4580B
CEmfPlusEnumState::ExtTextOutA(void)                                                                                        25D47AA6
CEmfPlusEnumState::ExtTextOutW(void)                                                                                        25D47A4E
```

Line 247 of 9411

Let’s have a look at some specific bugs.

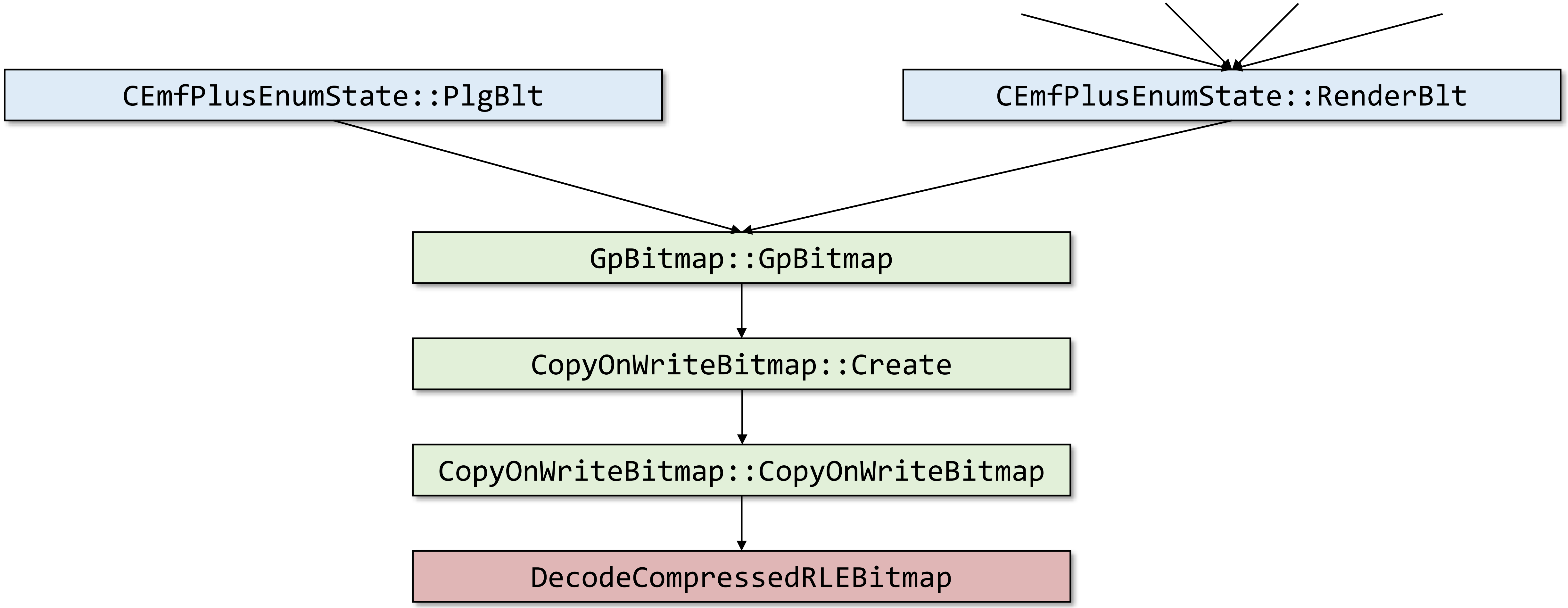
## CVE-2016-3301

| | |
|---|---|
| **Impact:** | Write-what-where |
| **Record:** | All records operating on DIBs |
| **Exploitable in:** | Microsoft Office |
| **CVE:** | CVE-2016-3301 |
| ***google-security-research* entry:** | 824 |
| **Fixed:** | MS16-097, 9 August 2016 |

## RLE-compressed bitmaps in EMFs

- As previously mentioned, multiple EMF records include DIBs.
- DIBs can be compressed with simple schemes, such as 4- and 8-bit *Run Length Encoding*.
  - Denoted by the `biCompression` field in the headers.
- When reading through the code of some handlers, I discovered that 8-bit RLE is supported in GDI+.
- RLE decompression has historically been a very frequent source of bugs.

# Reaching the code


CEmfPlusEnumState::PlgBlt
CEmfPlusEnumState::RenderBlt
GpBitmap::GpBitmap
CopyOnWriteBitmap::Create
CopyOnWriteBitmap::CopyOnWriteBitmap
DecodeCompressedRLEBitmap

## Inside `DecodeCompressedRLEBitmap()`

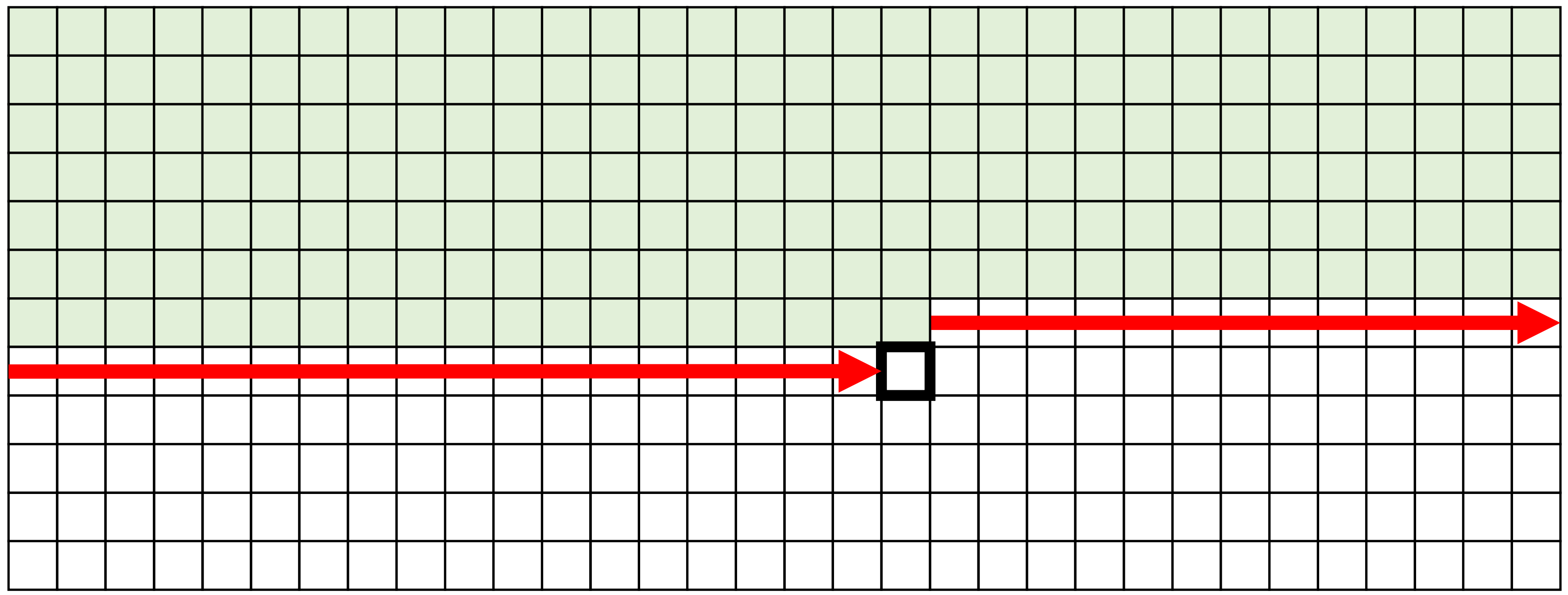
- Two values are calculated:

```
columns = abs(biHeight)

bytes_per_row = abs(biWidth * (((biPlanes * biBitCount + 31) & 0xFFFFFFE0) / 8))
```

- The output buffer is allocated from the heap with size `columns * bytes_per_row`.
  - High degree of control over the buffer length.
- Interpretation and execution of the RLE „program” begins.

## „End of Line” opcode

- Moves the output pointer to the next line (at the same offset).



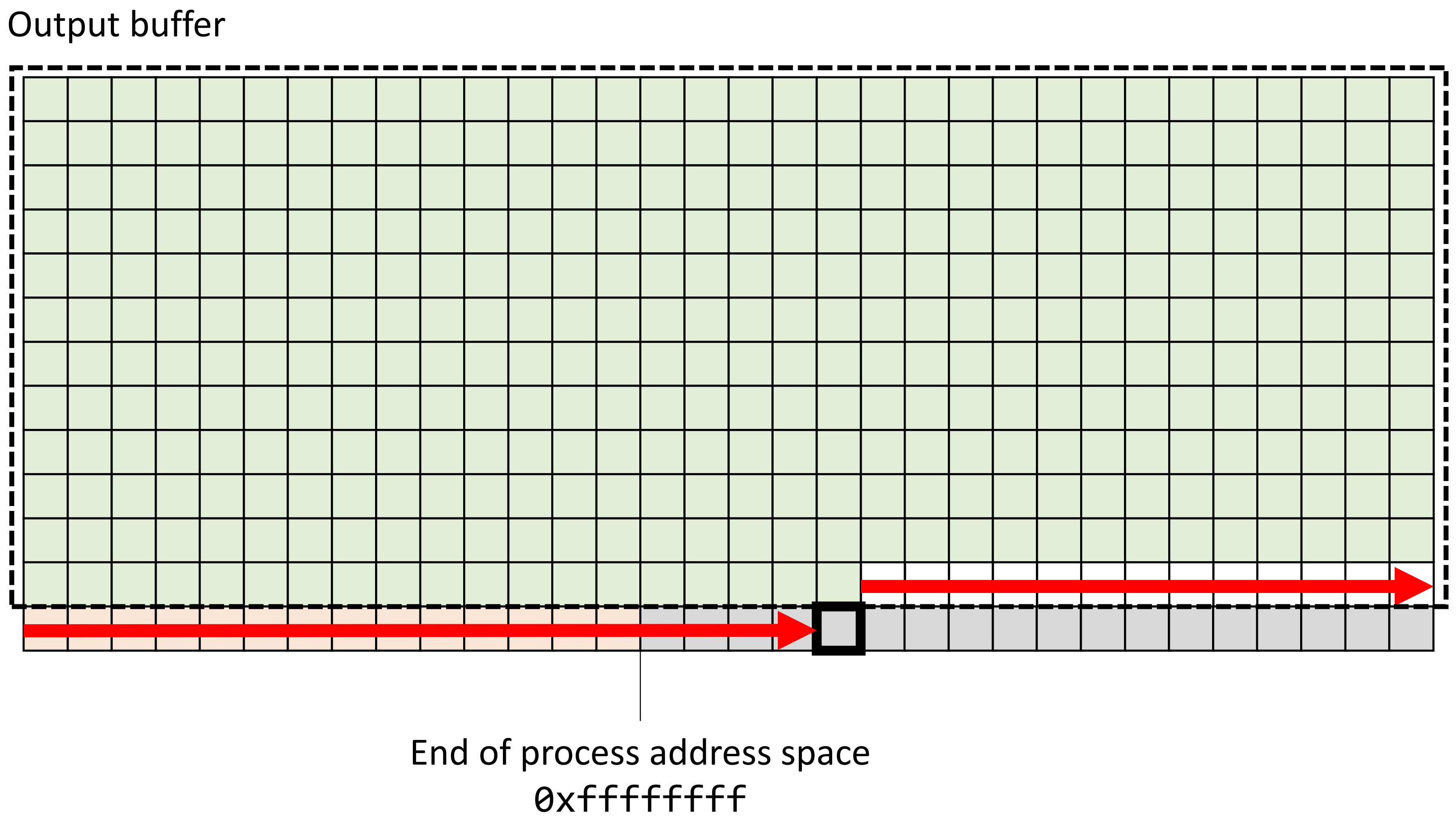
## „End of Line” opcode

- In GDI+, implemented as follows:

```
out_ptr += bytes_per_row;
if (out_ptr > output_buffer_end) {
  // Bail out.
}
```

- Bounds checking implemented to prevent any kind of out-of-bounds access.
- Happens to work correctly on 64-bit platforms, but is the condition really sufficient?

# Insufficient validation



## Tricky pointer arithmetic

- For very wide bitmaps, the distance from the current output pointer to the end of the address space can be smaller than the scanline width.
- The expression:

```
out_ptr += bytes_per_row;
```

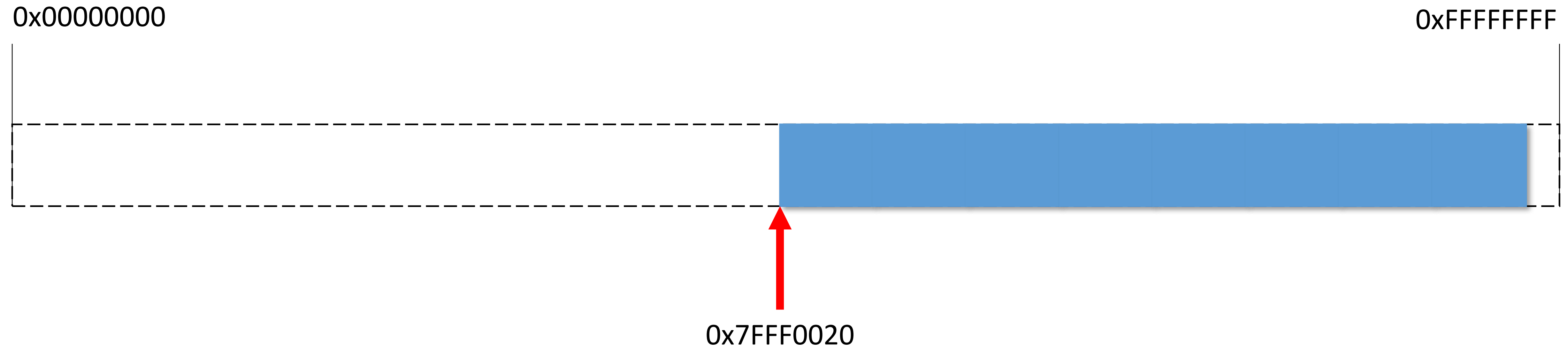
can overflow, which will cause the subsequent check to have no effect.

- As a result, it is possible to set the output pointer to a largely controlled address.

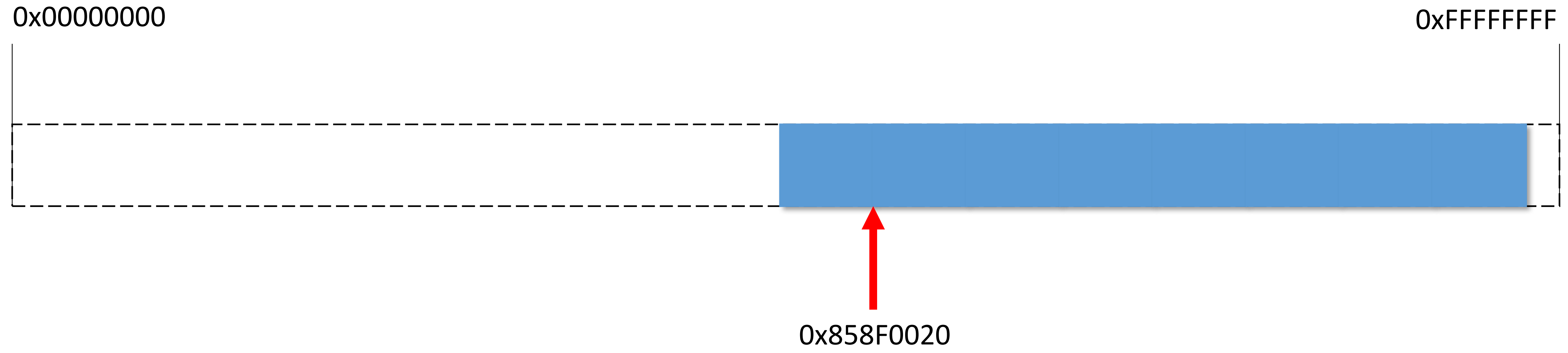
## Example

- `biWidth` = **0x05900000**
- `biHeight` = **0x00000017**
- `biPlanes` = **0x0001**
- `biBitCount` = **0x0008**
- As a result, `columns` = **0x17** and `bytes_per_row` = **0x590000**.
- Total buffer `size` = **0x7FF00000** (almost 2 GB).
- Example allocation address: **0x7FFFF0020**, end: **0xFFEF0020**.

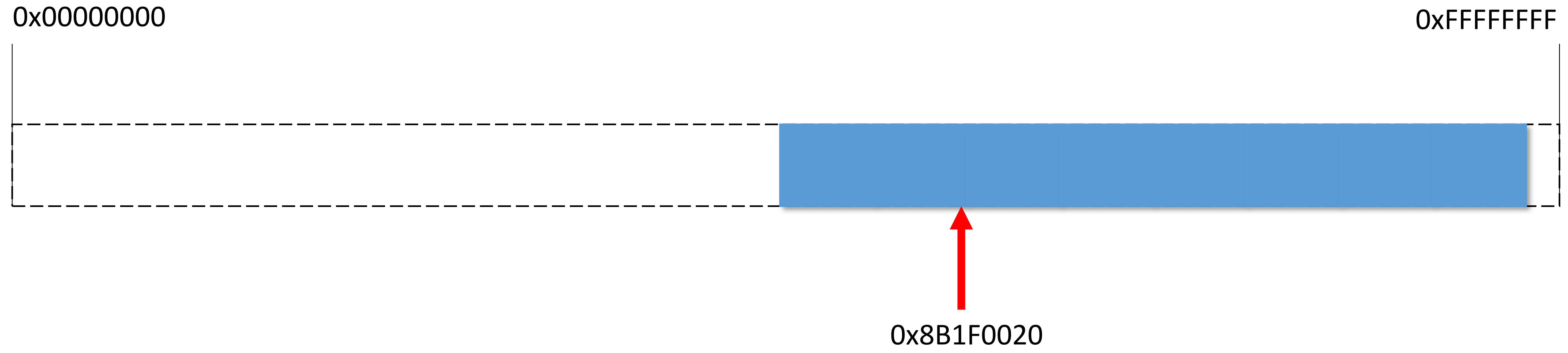
# Memory address space layout



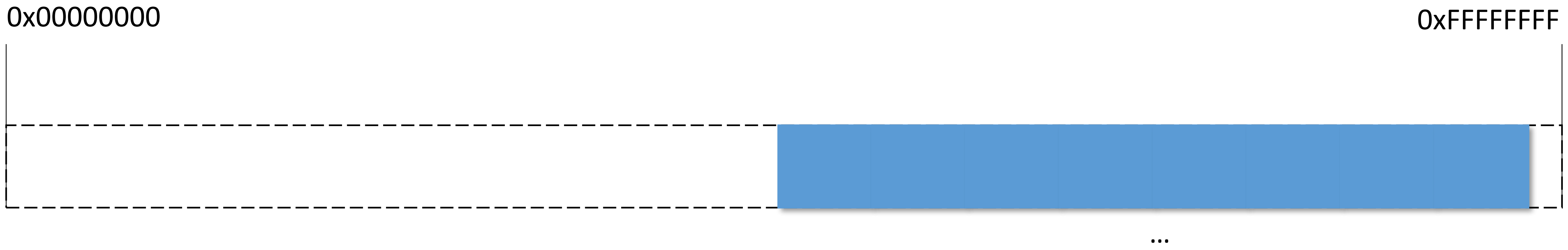
# Memory address space layout (EOL #1)



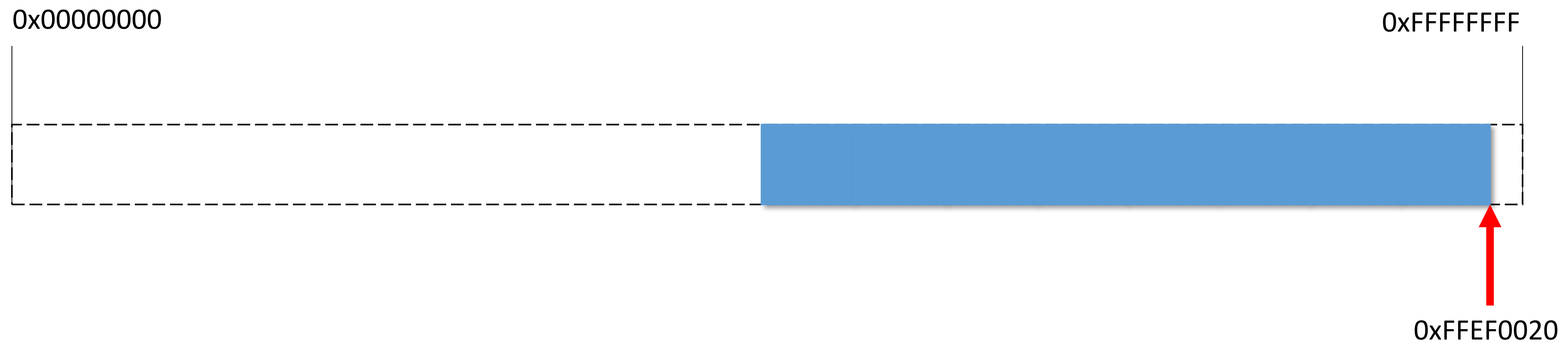
## Memory address space layout (EOL #2)



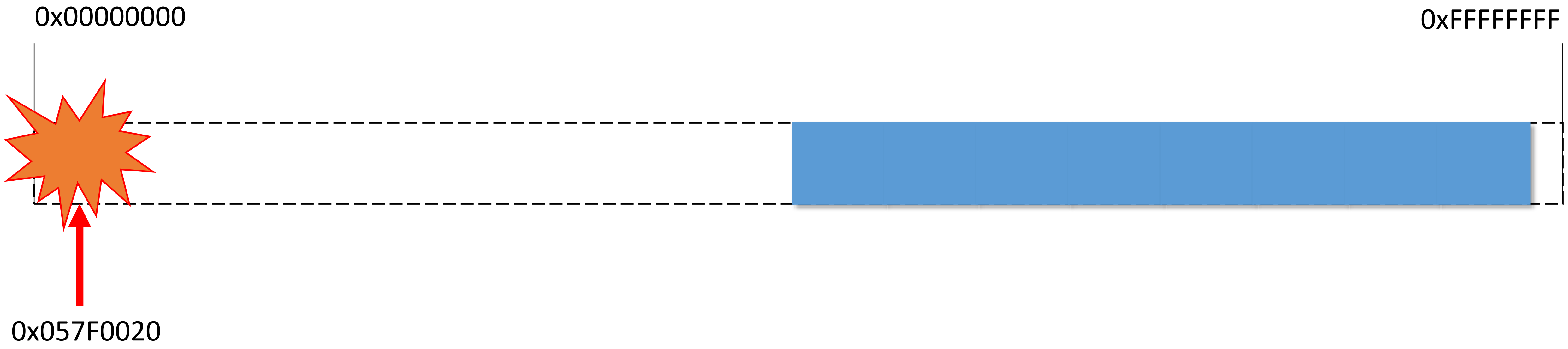
## Memory address space layout (EOL #3-22)



## Memory address space layout (EOL #23)



# Memory address space layout (EOL #24)



```
(3434.194): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0011015e ebx=ffef0020 ecx=000000fe edx=057f01cc esi=057f0020 edi=0011a6f0
eip=6b090e5a esp=0037f290 ebp=0037f2ac iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010207
gdiplus!DecodeCompressedRLEBitmap+0x195:
6b090e5a 8816            mov     byte ptr [esi],dl          ds:002b:057f0020=??
0:000> kb
ChildEBP RetAddr  Args to Child
0037f2ac 6b091124 057f0020 cc11012c 0037f2cc gdiplus!DecodeCompressedRLEBitmap+0x195
0037f6f4 6b092c7a 001100f8 0011012c 00000000 gdiplus!CopyOnWriteBitmap::CopyOnWriteBitmap+0x96
0037f708 6b0932cc 001100f8 0011012c 00000000 gdiplus!CopyOnWriteBitmap::Create+0x23
0037f720 6b0c1e8b 001100f8 0011012c 00000000 gdiplus!GpBitmap::GpBitmap+0x32
0037f804 6b0c7ed1 0000004f 00143a30 0000a67c gdiplus!CEmfPlusEnumState::PlgBlt+0x92
…
```

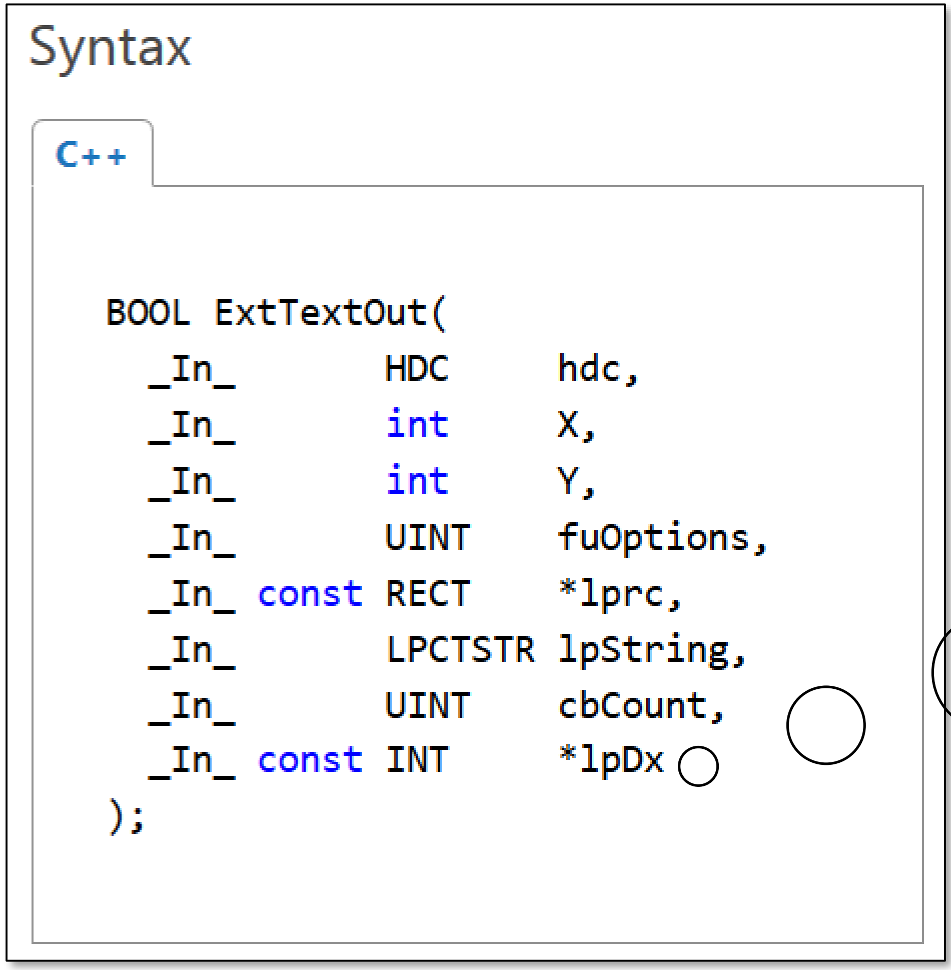
## Summary

- Requirement: 32-bit process with PAE enabled.
  - Full 4 GB address space must be available to the program.
- Outcome: a write-what-where condition, with a very high degree of control over the „where”.
  - Besides achieving a specific value, the overwritten region must also be below the original output buffer.
- Exploitation reliability highly depends on the state of the address space at the time of loading the image.

# CVE-2016-3304

| | |
|---|---|
| **Impact:** | Heap-based buffer overflow |
| **Record:** | EMR_EXTTEXTOUTA, EMR_POLYTEXTOUTA |
| **Exploitable in:** | Microsoft Office |
| **CVE:** | CVE-2016-3304 |
| ***google-security-research* entry:** | 828 |
| **Fixed:** | MS16-097, 9 August 2016 |

# ExtTextOutA() and PolyTextOutA()



Syntax

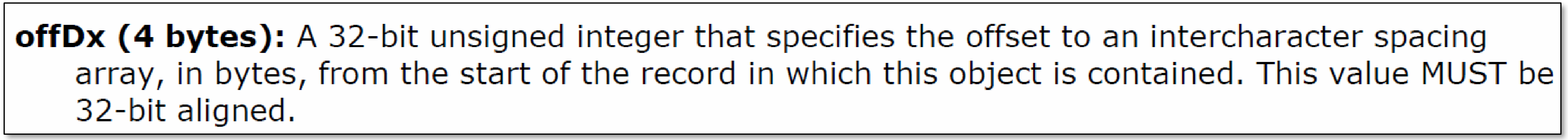
C++

```
BOOL ExtTextOut(
  _In_       HDC     hdc,
  _In_       int     X,
  _In_       int     Y,
  _In_       UINT    fuOptions,
  _In_ const RECT    *lprc,
  _In_       LPCTSTR lpString,
  _In_       UINT    cbCount,
  _In_ const INT     *lpDx
);
```

*lpDx* [in]

A pointer to an optional array of values that indicate the distance between origins of adjacent character cells. For example, lpDx[*i*] logical units separate the origins of character cell *i* and character cell *i* + 1.

## The Dx array in EMF records


**offDx (4 bytes):** A 32-bit unsigned integer that specifies the offset to an intercharacter spacing array, in bytes, from the start of the record in which this object is contained. This value MUST be 32-bit aligned.

## Trivial bug in the function

- The Dx array is supposed to have N elements, where N is the number of characters being displayed.
- Below is the record size validation check:

```
if (record_size - offString >= nChars &&
    (!nChars || record_size - 4 >= record->emrtext.offDx)) {
  // Validation passed, continue processing the record.
}
```

- See anything missing?

## Trivial bug in the function

- The code checks that the Dx array may hold 4 bytes.
  - What should really be verified is if it can hold 4 × N bytes.
  - Typical human error in the sanity check.
- So what? This should only lead to an out-of-bounds read, since it’s a problem with input buffer validation, right?
  - Yes, if not for the extra logic later in the code.

## Extended function logic

- Attempt to convert the string to wide-char, using `MultiByteToWideChar()`.
  - The code page is the one specified in the most recently selected font.
- If all characters are converted, `CEmfPlusEnumState::PlayExtTextOut()` is called as normal.
- But otherwise…

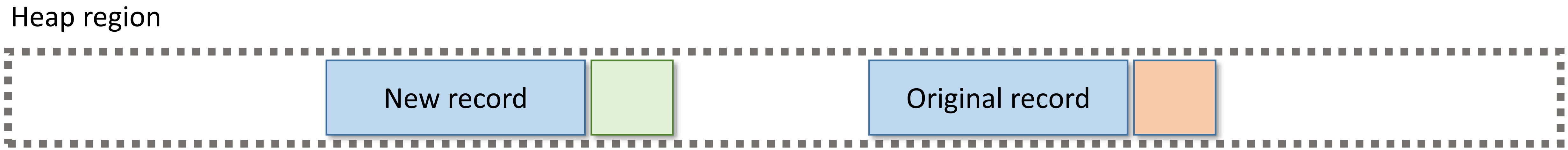
## DBCS (Double-byte character sets) handling

- Basically means representing characters by means of more than 1 byte in certain encodings which support it.
- The handling is implemented as follows:
  - An exact copy of the EMF record is allocated (of the same size).
  - Dx array items are rewritten from the original record to the new one, ommitting entries for „lead bytes” (`IsDBCSLeadByteEx()` returns TRUE).
  - The new record is processed normally from now on.

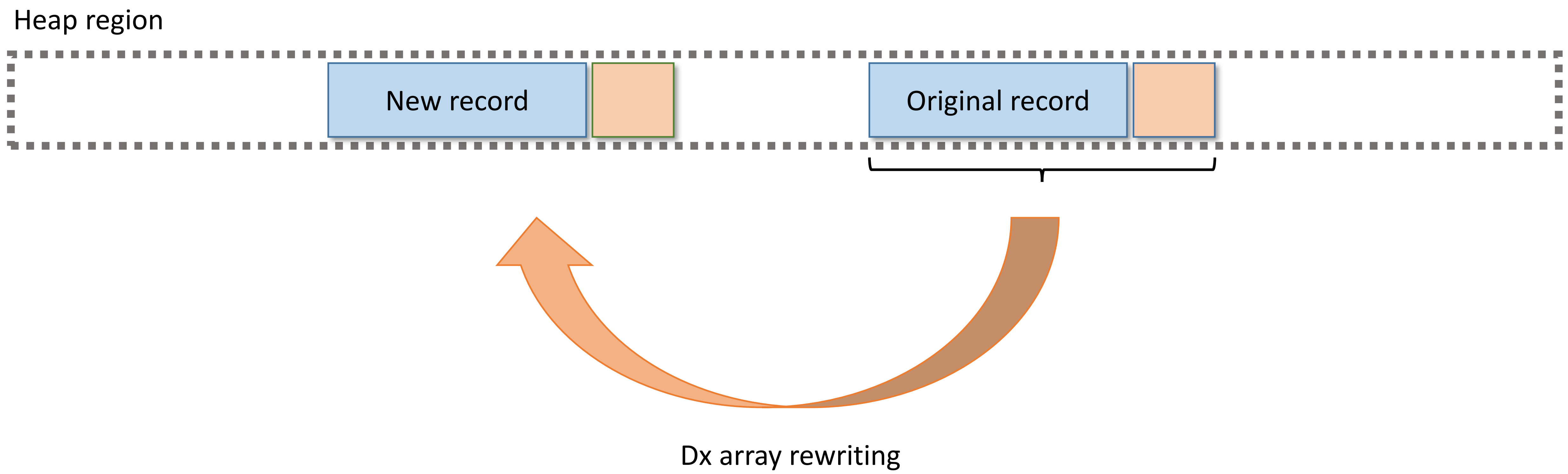
## Reaching the code path

- A font with a code page supporting DBCS must be selected first.
  - Typically CJK (Chinese, Japanese, Korean) code pages, e.g. `SHIFTJIS_CHARSET`.
  - Then, one of the affected records must be used, including at least one „lead byte”.
- The outcome is a typical heap-based buffer overflow, with data read from beyond the bounds of another allocation.
  - With some heap massaging, this should allow for a mostly controlled overwrite.

## Heap overflow scheme



## Heap overflow scheme



```
(2a8c.2bd8): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=772336ab edx=0022cb85 esi=03bd0000 edi=1171ffc0
eip=7728e815 esp=0022cdd8 ebp=0022ce50 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00200206
ntdll!RtlReportCriticalFailure+0x29:
7728e815 cc              int     3
0:000> kb
ChildEBP RetAddr  Args to Child
0022ce50 7728f749 c0000374 772c4270 0022ce94 ntdll!RtlReportCriticalFailure+0x29
0022ce60 7728f829 00000002 64dc1326 03bd0000 ntdll!RtlpReportHeapFailure+0x21
0022ce94 7724ab46 0000000c 03bd0000 1171ffc0 ntdll!RtlpLogHeapFailure+0xa1
0022cf84 771f3431 00000258 00000260 03bd00c4 ntdll!RtlpAllocateHeap+0x7b2
0022d008 695071ec 03bd0000 00000000 00000258 ntdll!RtlAllocateHeap+0x23a
0022d01c 6951bbf1 00000258 116b5104 03bdd558 gdiplus!GpMalloc+0x16
0022d030 69557185 116b50e0 116b50e0 03bdd558 gdiplus!GpGraphics::Save+0x11
0022d4b0 69557bdc 116b50e0 116b5104 116b30d8 gdiplus!CEmfPlusEnumState::PlayExtTextOut+0xda
0022d4ec 69557f25 00000053 03bdae00 00006044 gdiplus!CEmfPlusEnumState::ExtTextOutA+0x136
0022d500 695286ca 00000053 00006044 0d67b568 gdiplus!CEmfPlusEnumState::ProcessRecord+0x13b
0022d51c 69528862 00000053 00000000 00006044 gdiplus!GdipPlayMetafileRecordCallback+0x6c
0022d544 768155f4 9d211b17 0d567180 0d67b568 gdiplus!EnumEmfDownLevel+0x6e
0022d5d0 6952aa36 9d211b17 403581b3 695287f4 GDI32!bInternalPlayEMF+0x6a3
```

## GDI+ information disclosure bugs

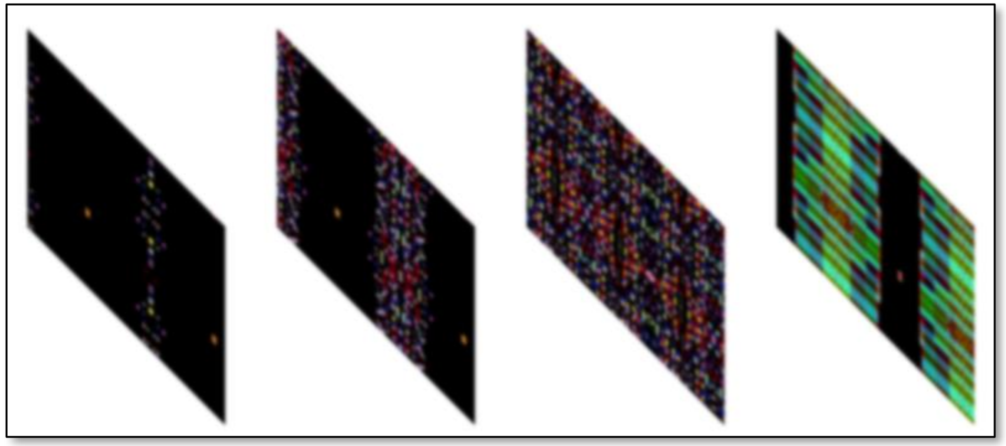
| | |
|---|---|
| **Impact:** | Heap memory disclosure |
| **Record:** | All records handling DIBs |
| **Exploitable in:** | Microsoft Office Online |
| **CVE:** | CVE-2016-3262, CVE-2016-3263 |
| ***google-security-research* entry:** | 825, 829 |
| **Fixed:** | MS16-120, 11 October 2016 |

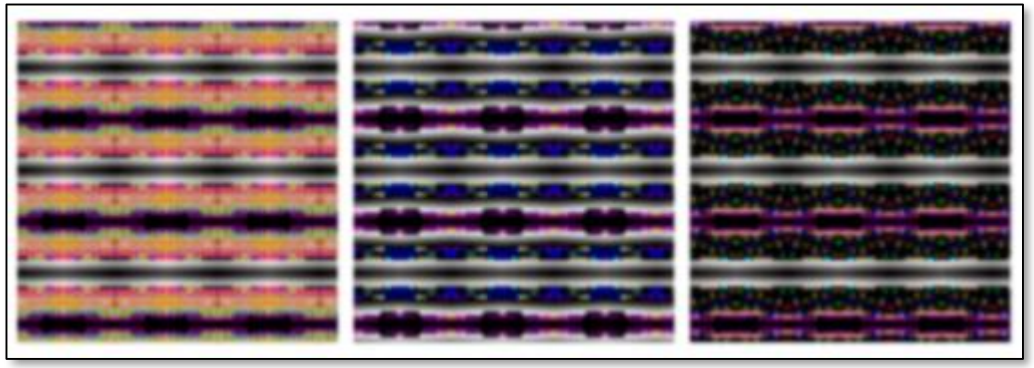
## GDI+ versus DIB

- Not unlike GDI, GDI+ didn’t avoid information disclosure bugs related to the handling of bitmaps.
- Specifically:
  1. If the data stream of a RLE-compressed bitmap begins with an „End of bitmap” marker, the entirety of the image’s output buffer remains uninitialized (contains junk heap data).
  2. No checks are performed to ensure that the bitmap palette fits entirely within the EMF record.

## Bugs clearly visible

- When loading proof-of-concept pictures into Word, it’s clearly visible that junk data is displayed as pixels.





## Remote exploitability?

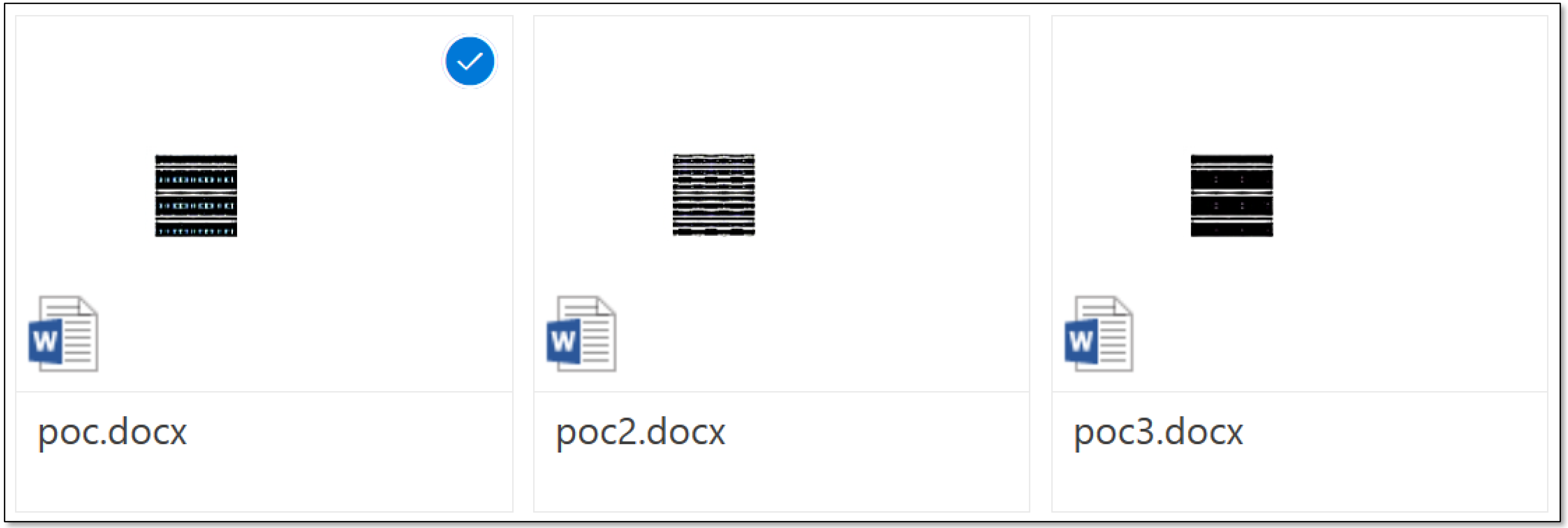
- Displaying heap memory is not a serious condition if the pixels cannot be retrieved back somehow.
- The only obvious targets for the bugs are Office programs, where no interaction is available.
- Still reported to Microsoft to get their view on severity and possible exploitation paths.
- MSRC closed out the issues as „vNext" (won't be patched in a bulletin, candidate for a next-version fix).

## Severity assessment

- I agreed with the decision, as it was in line with my own understanding of the exposure.
  - P0 bugs #825 and #829 were derestricted on July 26 and August 9, respectively.
- At the beginning of August, Ivan Fratric mentioned during a chat that GDI+/EMF bugs may also be exploitable remotely, in Office Online.
  - I had no idea the program even existed.
  - Especially interesting for GDI+ memory disclosure bugs, which are not otherwise exploitable.
  - EMF images cannot be inserted into documents, but existing .docx with embedded EMF can.

## Office Online

- I verified this a few weeks later, and…



# Office Online

- The EMF images were rendered differently each time.
- Apparent remote memory disclosure from the renderer process on Microsoft’s servers.
- Sent the new information to MSRC for reconsideration.
- They admitted the Office Online scenario had not been considered before, and it makes the bugs fix-worthy.
- They should have been fixed, as per the October Patch Tuesday.

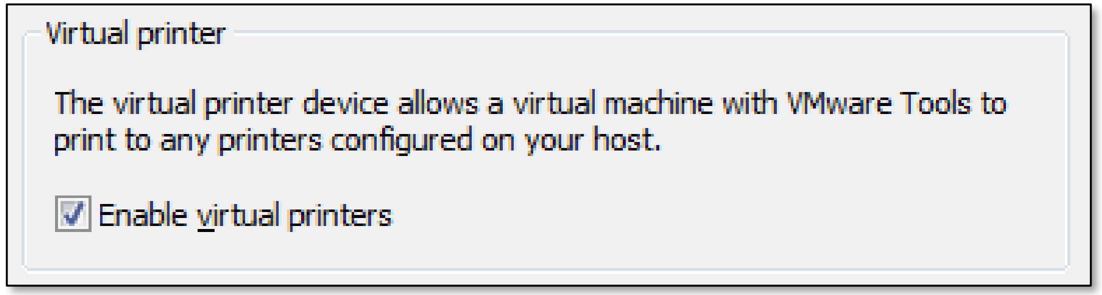
## Hacking VMware Workstation

## EMF in print spooling

- EMF files are also used heavily in print spooling.
- This opens up more format-related attacks vectors, in the form of printer drivers (and other related software).
- One such feasible target is VMware Workstation.

## Virtual printers

- A feature which allows a virtual machine to print documents to printers available on the host (basically printer sharing).
- A feasible VM escape attack vector.
- To my best knowledge, it was enabled by default in 2015, but it’s no longer the case (likely thanks to bugs reported by Kostya Kortchinsky).
- Still a frequently used option.

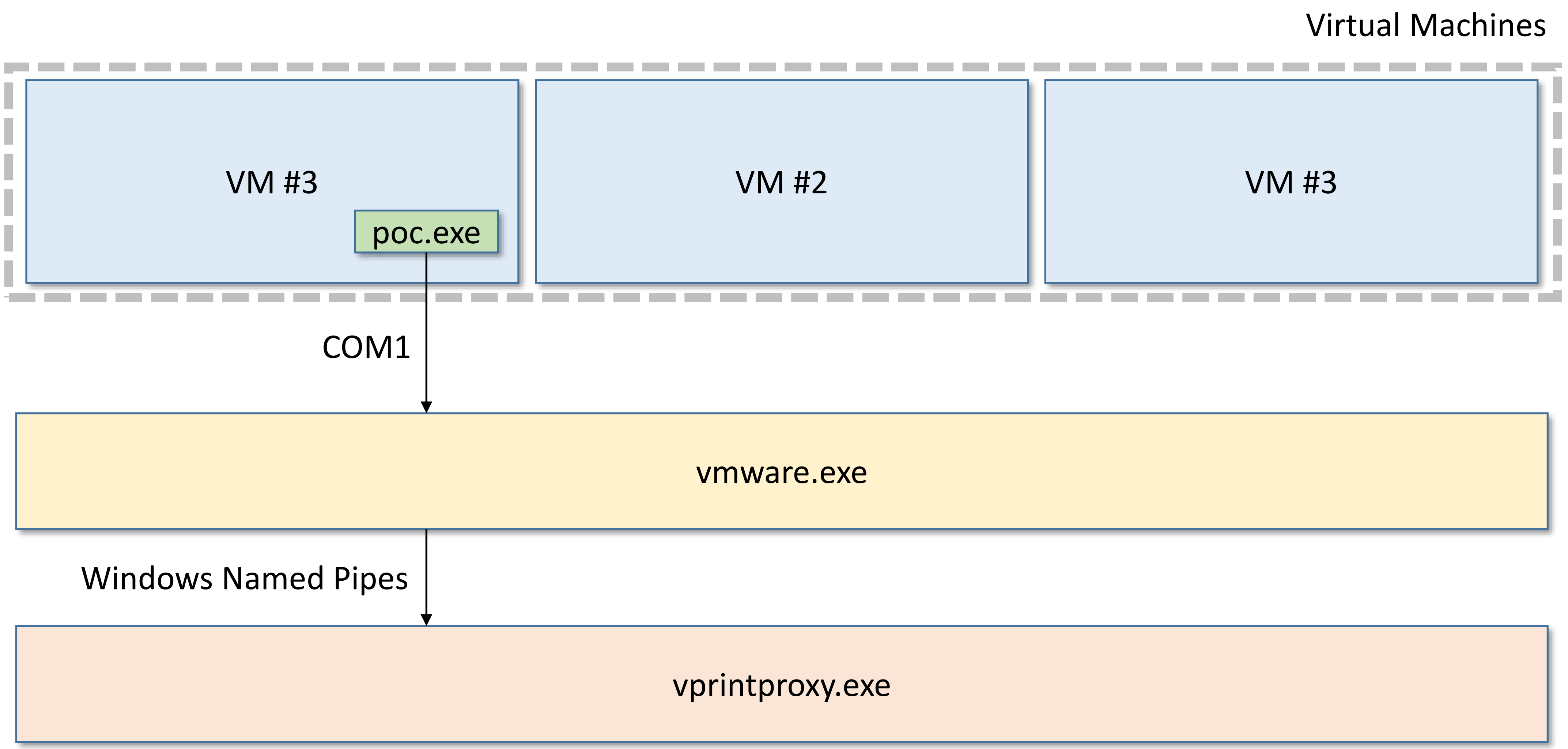


Virtual printer

The virtual printer device allows a virtual machine with VMware Tools to print to any printers configured on your host.

[✓] Enable virtual printers

# Architecture



## Architecture

- The attacked process is vprintproxy.exe running on the host.
  - Receives almost verbatim data sent by an unprivileged process in a guest system.
  - Quite a communication channel.
- The data is sent in the form of EMFSPOOL files.
  - Similar to EMF, with the extra option to embed fonts in various formats.

## TPView

- More specifically, the most interesting EMF handling takes place in TPview.dll.
  - Together with some other printer-related libraries, they all seem to be developed by a third party, ThinPrint.
- Mostly just falls back to GDI, but also performs specialized handling of several record types.
  - Used to be full of simple bugs, but Kostya found (nearly) all of them!
  - Took another look, discovered a double-free and out-of-bounds `memset()`, but that’s all (issues #848 and #849).

## JPEG2000 decoding

- There was one last custom EMF record which seemed completely unexplored.
  - ID = 0x8000.
  - Based on debug strings, it was clear that it was related to JPEG2000 decoding.
- I am no expert at JPEG2K, and the code doesn’t seem to be convenient for manual auditing.
- Let’s fuzz it?

## Approaching the fuzzing

- Best fuzzing: on Linux, at scale, with AddressSanitizer and coverage feedback.
- After some research, it turns out that the JPEG2000 decoder is authored by yet another vendor, LuraTech.
    - Commercial license, source code not freely available.
- So, are we stuck with TPview.dll wrapped by VMware Workstation?
    - Still feasible, but more complex, slower, and less advanced.

# More research

- After some more digging, I found out that the same vendor released a freeware JPEG2000 decoding plugin for the popular IrfanView program.
  - JPEG2000.DLL.
  - Cursory analysis shows that this is the same or a very similar code base.
- The plugin interface is an extremely simple to use, and resembles the following definition.

```
HGLOBAL ReadJPG2000(IN PCHAR lpFilename,
                    IN DWORD dwUnknown,
                    OUT PCHAR lpStatus,
                    OUT PCHAR lpFormat,
                    OUT LPDWORD lpWidth,
                    OUT LPDWORD lpHeight);
```

## Getting there...

- Thanks to this, we can already quickly fuzz-test the implementation in a single process on Windows, without running VMware at all.
  - A wrapper program for loading the DLL and calling the relevant function is <50 LOC long.
- However, I'd really prefer to have this on Linux...

## Fuzzing DLL on Linux

- Why not, really?
- The preferred base address is 0x10000000, which is available in the address space.
  - Relocations not required; sections must be mapped with respective access rights.
- Other actions:
  - Resolve necessary imports.
  - Obtain the address of the exported function.
  - Call it to execute the decoding.
- Should work!

## Resolving imports

- The Import Table may be the only troublesome part.
  - WinAPI functions not available on Linux.
- The DLL imports from ADVAPI32, KERNEL32, MSVCRT, SHELL32 and USER32.
  - C Runtime imports can be directly redirected to libc.
  - All the other ones would have to be rewritten or at least stubbed-out.

# KERNEL32 imports

- Three WinAPI functions used in decoding: `GlobalAlloc`, `GlobalLock` and `GlobalUnlock`:

```
void *GlobalAlloc(uint32_t uFlags, uint32_t dwBytes) __attribute__((stdcall));
void *GlobalAlloc(uint32_t uFlags, uint32_t dwBytes) {
  void *ret = malloc(dwBytes);
  if (ret != NULL) {
    memset(ret, 0, dwBytes);
  }
  return ret;
}

void *GlobalLock(void *hMem) __attribute__((stdcall));
void *GlobalLock(void *hMem) {
  return hMem;
}

bool GlobalUnlock(void *hMem) __attribute__((stdcall));
bool GlobalUnlock(void *hMem) {
  return true;
}
```

## Missing libc imports

- Two MSVCRT-specific imports were found, which had to be reimplemented:

```
long long _ftol(double val) __attribute__((cdecl));
long long _ftol(double val) {
  return (long long)val;
}

double _CIpow(double x, double y) __attribute__((cdecl));
double _CIpow(double x, double y) {
  return pow(x, y);
}
```

## It works!

```
$ ./loader JPEG2000.dll test.jp2
[+] Successfully loaded image (9b74ba8), format: 
JPEG2000 - Wavelet, width: 4, height: 4
```

## Running the fuzzing

- An internally available JPEG2000 input file corpus was used.
- The mutation strategy was adjusted to hit the 50/50 success/failure rate.
- Left the dumb fuzzer running for a few days, and...
  - ... 186 crashes with unique stack traces were found.

## Crash reproduction

- Keep in mind the crashes are still in the plugin DLL, not VMware Workstation.
- vprintproxy.exe is very convenient to use: creates a named pipe and reads exactly the same data that is written to COM1.
  - Once again we can check testcases without starting up any actual VMs.
- PageHeap enabled for better bug detection and deduplication.

## Final results

| Instruction | Reason |
|---|---|
| add [eax+edx*4], edi | Heap buffer overflow |
| cmp [eax+0x440], ebx | Heap out-of-bounds read |
| cmp [eax+0x8], esi | Heap out-of-bounds read |
| cmp [edi+0x70], ebx | Heap out-of-bounds read |
| cmp [edi], edx | Heap out-of-bounds read |
| cmp dword [eax+ebx*4], 0x0 | Heap out-of-bounds read |
| cmp dword [esi+eax*4], 0x0 | Heap out-of-bounds read |
| div dword [ebp-0x24] | Division by zero |
| div dword [ebp-0x28] | Division by zero |
| fld dword [edi] | NULL pointer dereference |
| idiv ebx | Division by zero |
| idiv edi | Division by zero |
| imul ebx, [edx+eax+0x468] | Heap out-of-bounds read |
| mov [eax-0x4], edx | Heap buffer overflow |
| mov [ebx+edx*8], eax | Heap buffer overflow |
| mov [ecx+edx], eax | Heap buffer overflow |
| mov al, [esi] | Heap out-of-bounds read |
| mov bx, [eax] | NULL pointer dereference |
| mov eax, [ecx] | NULL pointer dereference |
| mov eax, [edi+ecx+0x7c] | Heap out-of-bounds read |

| Instruction | Reason |
|---|---|
| mov eax, [edx+0x7c] | Heap out-of-bounds read |
| movdqa [edi], xmm0 | Heap buffer overflow |
| movq mm0, [eax] | NULL pointer dereference |
| movq mm1, [ebx] | NULL pointer dereference |
| movq mm2, [edx] | NULL pointer dereference |
| movzx eax, byte [ecx-0x1] | Heap out-of-bounds read |
| movzx eax, byte [edx-0x1] | Heap out-of-bounds read |
| movzx ebx, byte [eax+ecx] | Heap out-of-bounds read |
| movzx ecx, byte [esi+0x1] | Heap out-of-bounds read |
| movzx ecx, byte [esi] | Heap out-of-bounds read |
| movzx edi, word [ecx] | NULL pointer dereference |
| movzx esi, word [edx] | NULL pointer dereference |
| push dword [ebp-0x8] | Stack overflow (deep / infinite recursion) |
| push ebp | Stack overflow (deep / infinite recursion) |
| push ebx | Stack overflow (deep / infinite recursion) |
| push ecx | Stack overflow (deep / infinite recursion) |
| push edi | Stack overflow (deep / infinite recursion) |
| push esi | Stack overflow (deep / infinite recursion) |
| rep movsd | Heap buffer overflow, Heap out-of-bounds read |

## Final results

- Crashes at **39** unique instructions.
  - Many occurring at various points of generic functions such as `memcpy()`, so not the most accurate metric.
  - Quick classification: **18** low severity, **15** medium severity, **6** high severity.
- All reported to VMware on June 15.
- Fixed as part of VMSA-2016-0014 on September 13 (within 90 days).

## Closing thoughts

## Closing thoughts

- Metafiles are complex and interesting files, certainly worth researching further.
  - Supported by a variety of valid attack vectors.
- They can even teach you things about the system API (i.e. the NamedEscape interface).
- As usual, the older and more obscure the format/implementation – the better for the bughunter.
- Inspiration with prior work pays off again.
- The right tool for the right job – manual code auditing vs fuzzing.

# Thanks!

@j00ru

http://j00ru.vexillium.org/

j00ru.vx@gmail.com