Graficzny format Windows Metafile

Analiza wektorów ataku i najnowszych podatności

Mateusz “j00ru” Jurczyk

Security PWNing Conference, Warszawa 2016

PS> whoami

- Project Zero @ Google
- Wicekapitan @ Dragon Sector
- Badacz zainteresowany bezpieczeństwem niskopoziomowym, odkrywaniem błędów i ich eksploitacją.
- http://j00ru.vexillium.org/
- @j00ru

Agenda

- Wstęp do plików Windows Metafile, architektury GDI, wektorów ataku.
- Hacking:
  - Internet Explorer (GDI)
  - Windows Kernel (ATMFD.DLL)
  - Microsoft Office (GDI+)
  - VMware Workstation (Print Spooling)
- Podsumowanie.

Wstęp do Windows GDI & Metafiles

Windows GDI

- *Graphics Device Interface*.
- Umożliwia aplikacjom trybu użytkownika operowanie w trybie graficznym (malowanie kształtów, wypisywanie tekstu) na monitorach i drukarkach.
- Spora część systemowego API (prawie 300 udokumentowanych funkcji).
- Obecny w systemie od samego początku (Windows 1.0 wydany w 1985 r.).
  - Jeden z najstarszych podsystemów, większość oryginalnego kodu wciąż jest w użyciu 31 lat później.
  - Przypadkowo (?) również jeden z najbardziej podatnych komponentów.

Rysowanie

1. Pobranie uchwytu do tzw. *Device Context* (HDC).
   - Reprezentuje kontener na graficzne ustawienia różnych obiektów (*pens*, *brushes*, *palettes* itd.).
   - Może być użyty do rysowania na ekranie (typowy scenariusz), „na papierze” (za pośrednictwem drukarki) lub do pliku metafile.
   - Najprostszy przykład:

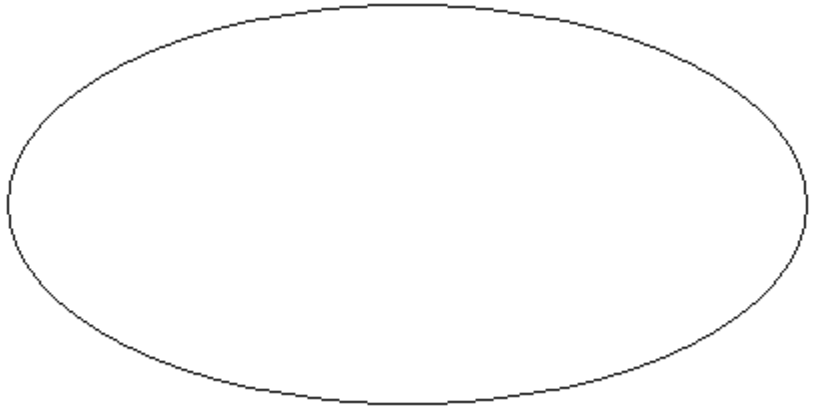
```
HDC hdc = GetDC(NULL);
```

(pobiera HDC dla całego ekranu)

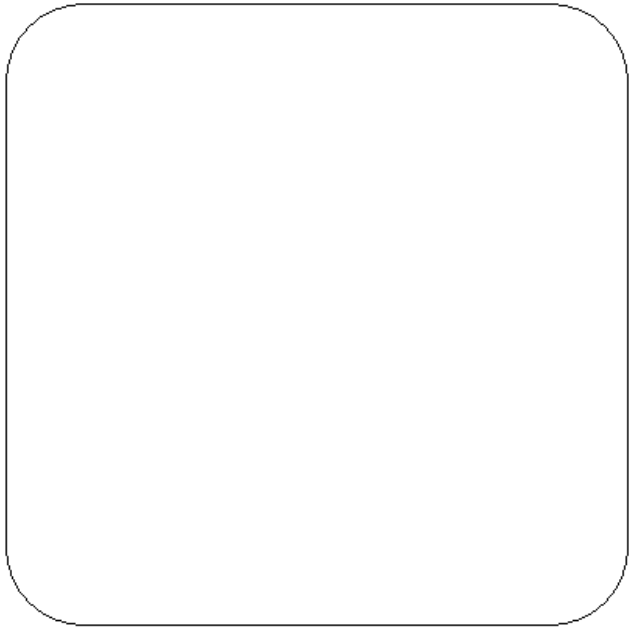
Rysowanie

2. Użycie funkcji rysującej.

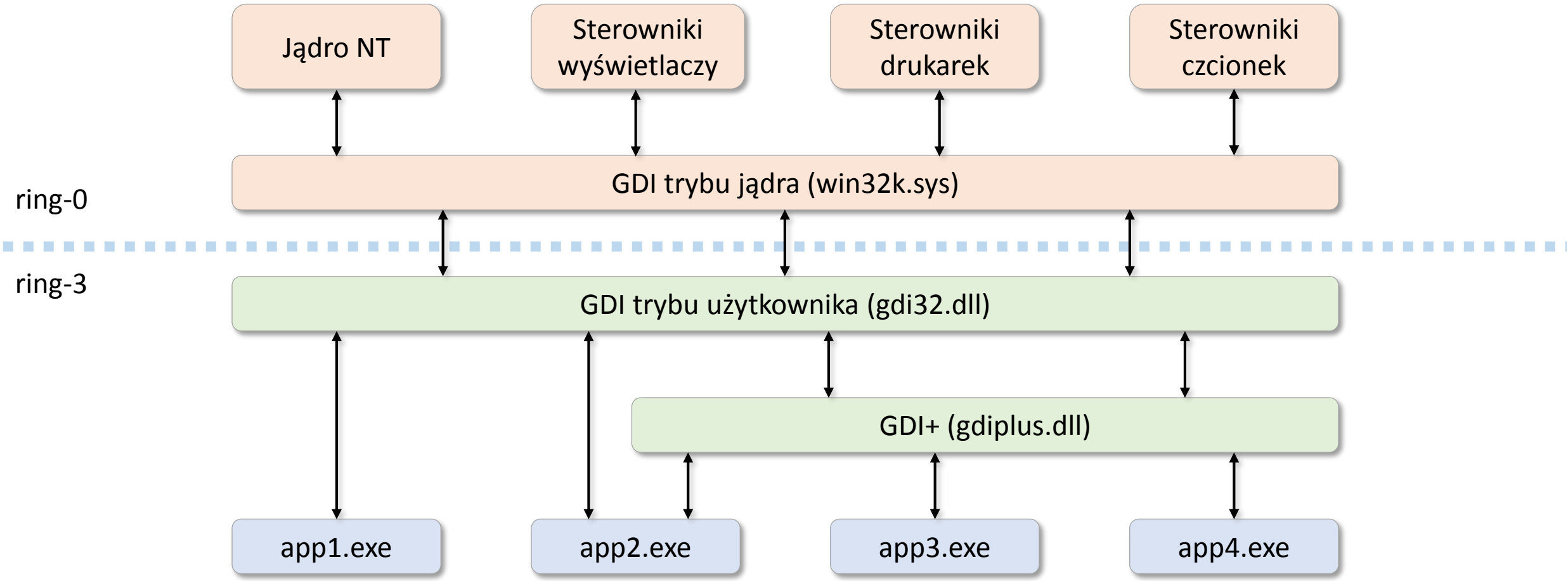
```
Ellipse(hdc, 100, 100, 500, 300);
```



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



Windows GDI – uproszczona architektura



Mapowania API trybu użytkownika i jądra

Większość funkcji GDI trybu użytkownika ma swoje bezpośrednie odpowiedniki po stronie jądra:

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

Windows Metafiles

Główna idea:

Przechowywanie obrazków jako list rekordów opisujących poszczególne wywołania funkcji GDI.

Windows Metafiles

- **Zalety:**
  - wymaga bardzo mało pracy ze strony obsługującego kodu, która sprowadza się do wywoływania odpowiednich funkcji GDI z ustalonymi parametrami.
  - zapewnia oficjalny sposób serializowania serii wywołań GDI do powtarzalnych obrazków.
  - może działać jako format wektorowy, rastrowy, lub oba naraz.
- **Wady:**
  - działa tylko na systemach Windows, o ile inny projekt nie stworzy kompletnej implementacji wszystkich funkcji GDI obsługiwanych przez pliki metafile.

Pierwsza wersja: WMF

- Oryginalne pliki metafile (WMF = **W**indows **M**eta**F**iles).
- Wprowadzone wraz z Windows 3.0 w 1990 r.
  - Nie są tak stare jak samo GDI, ale prawie.
- Początkowo udokumentowane w Windows 3.1 SDK (1994 r., tom 4).
  - Poprawiona, bardziej kompletna specyfikacja została wydana w 2006 r., i od tego czasu jest aktualizowana.
  - Opis wszystkich rekordów i struktur można znaleźć w dokumencie MS-WMF.

Pliki WMF – 60 wspieranych funkcji API

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

Niektóre z nich wyglądają ciekawie

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: to nie koniec!

- Format wspiera również pewne rekordy, które nie odpowiadają bezpośrednio wywołaniom GDI.
  - Nagłówek z metadanymi.
  - Osadzone pliki EMF.
  - Rekordy wchodzące w interakcję ze sterownikiem drukarki.
  - Marker końca pliku.
  - ...

WMF: to nie koniec!

- Większość interesujących rekordów można znaleźć w dwóch sekcjach:

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 thrc

Windows Metafile – przykład

```
...
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: bardzo przestarzały

- Pomimo dość wysokiego stopnia skomplikowania, format okazał się niezbyt dobrze przemyślany pod kątem nowoczesnych potrzeb.
- Wciąż wspierany przez GDI, a co za tym idzie, również niektóre z jego klientów (np. Microsoft Office, Paint, inne domyślne programy w systemie).
- Zasadniczo zapomniany w realnych zastosowaniach co najmniej od dekady.

WMF: niezalecany do użycia

- Sam Microsoft podaje wiele argumentów za jego odrzuceniem:

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).

Następny z kolei: EMF (**E**nhanced **M**eta**F**iles)

- Już w 1993 r. Microsoft wydał poprawioną wersję formatu, nazwaną EMF.
- Udokumentowany w oficjalnej specyfikacji MS-EMF.
- Przewyższa WMF pod wieloma względami:
  - używa 32-bitowej szerokości offsetów, w przeciwieństwie do poprzednich 16 bitów.
  - niezależny od urządzenia, na którym jest wyświetlany.
  - wspiera wiele nowych wywołań GDI, utrzymując kompatybilność wsteczną ze starymi rekordami.

Enhanced Metafile – przykład

```
...
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: interesujące rekordy

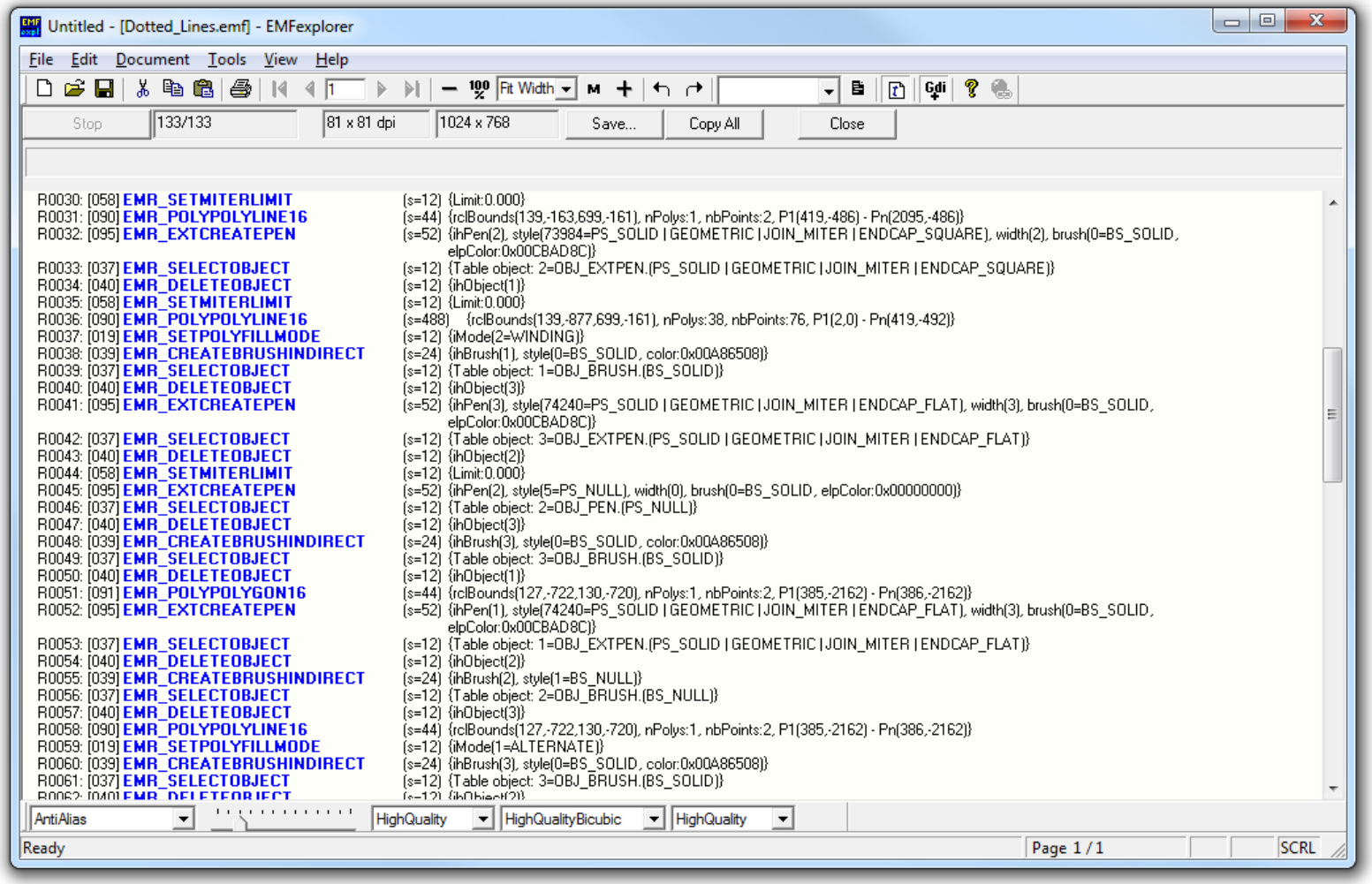
EMF: interesujące rekordy

EMF: interesujące rekordy

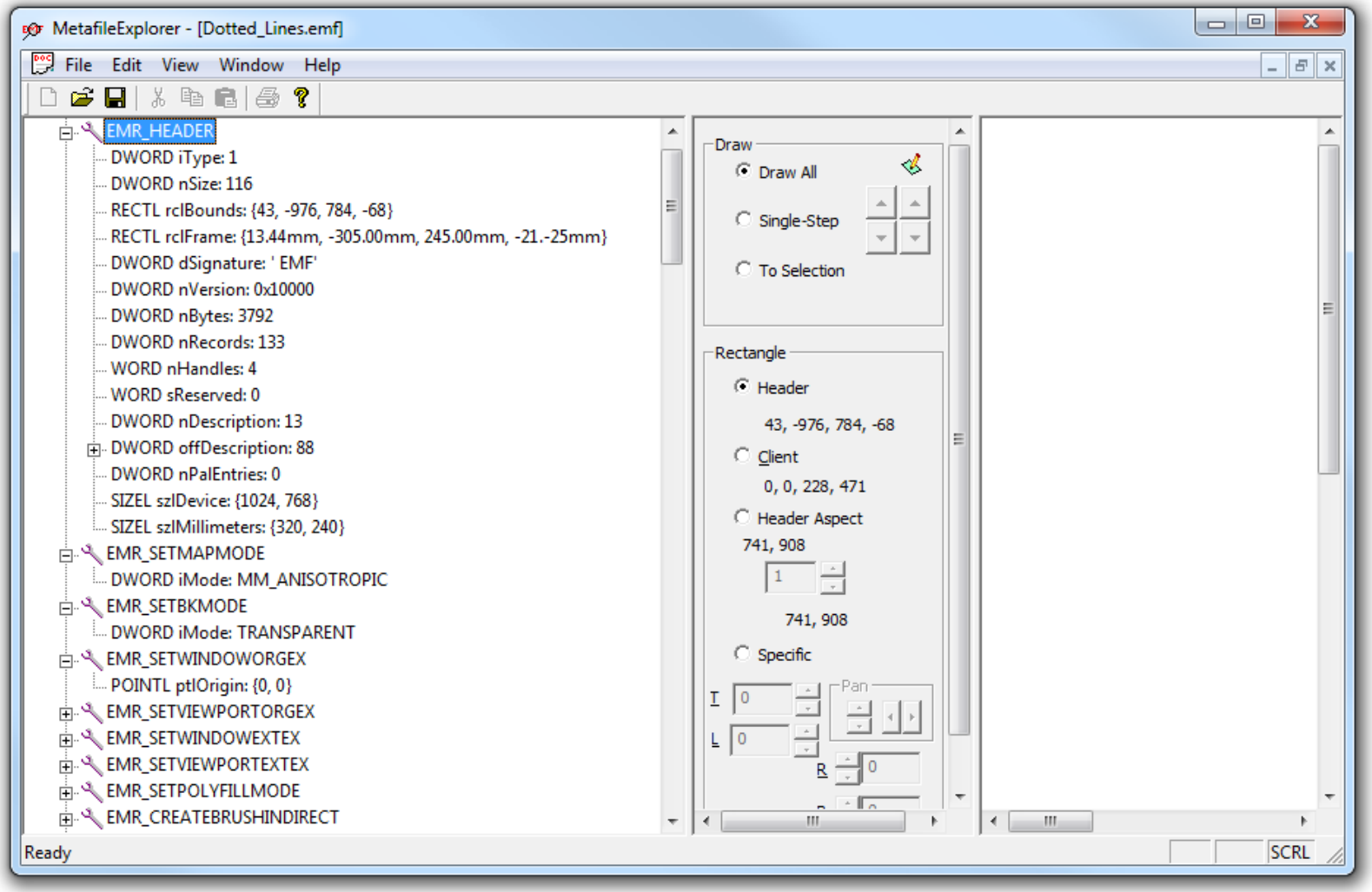
EMF: dzisiejsze wsparcie

- Pomimo tego, że jest tylko 3 lata młodszy od WMF, EMF pozostaje w aktywnym użyciu aż do dzisiaj.
  - Co prawda nie jako powszechny format graficzny, ale jako istotna płaszczyzna ataku.
- Wiele wektorów:
  - Programy używające Win32 GDI – w szczególności Internet Explorer.
  - Programy używające GDI+ – w szczególności Microsoft Office.
  - Sterowniki drukarek, włączając w to wsparcie dla drukarek w technologiach wirtualizacji.

Narzędzia – odczyt (EMFexplorer)



Narzędzia – odczyt (MetafileExplorer)



Narzędzia – odczyt i zapis (pyemf)

```
#!/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)
```

Najnowsza iteracja: EMF+

- GDI wspiera podstawowe operacje, ale brakuje mu wielu zaawansowanych opcji (anti-aliasing, koordynaty zmiennoprzecinkowe, obsługa JPEG/PNG itd.).
- Windows XP wprowadził w 2001 r. bardziej zaawansowaną bibliotekę o nazwie GDI+.
  - Technicznie moduł gdiplus.dll trybu użytkownika, oparty o GDI (gdi32.dll).
  - Zapewnia wysokopoziomowe interfejsy dla C++ i .NET.
  - GDI+ jest napisane w C++, więc dotyczą go wszelkie błędy związane z obsługą pamięci.

Najnowsza iteracja: EMF+

- Skoro jest nowy interfejs, to musi również istnieć nowy format obrazków z jego zserializowanymi wywołaniami.
- Powitajmy EMF+!
- W zasadzie to samo co EMF, ale reprezentujący wywołania GDI+.
- Dwa typy: EMF+ Only i EMF+ Dual.
  - „Only” zawiera wyłącznie rekordy GDI+ i może być wyświetlony tylko przez tę bibliotekę.
  - „Dual” opisuje obrazek dwoma zestawami rekordów, dzięki czemu może być wyświetlany przez programy oparte o GDI i GDI+.

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.

Formaty i implementacje w Windows

- W sumie trzy formaty do rozpatrzenia: WMF, EMF, EMF+.
- Trzy biblioteki: GDI, GDI+ i MF3216.
  - MF3216.DLL to biblioteka systemowa z jedną ważną eksportowaną funkcją: `ConvertEmfToWmf`.
  - Używana do automatycznej konwersji między WMF a EMF w schowku Windows.
    - „Synchronizowane” formaty `CF_METAFILEPICT` i `CF_ENHMETAFILE`.
  - Poszukiwania błędów bez rezultatów. ☹

Formaty i implementacje w Windows

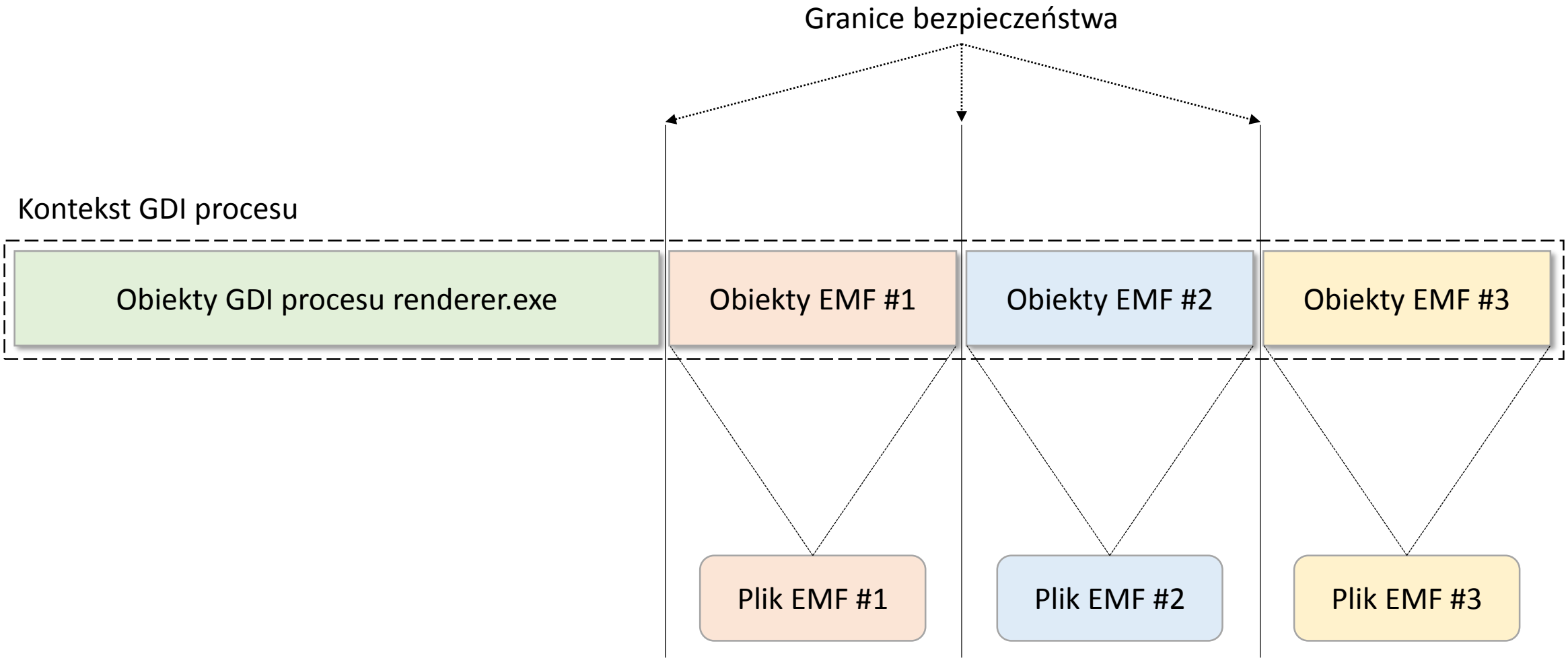
Biblioteka	Wspierane formaty
GDI	WMF, EMF
GDI+	WMF, EMF, EMF+
MF3216	EMF

W tej prelekcji skupimy się na wynikach audytu i exploitacji EMF, gdzie podczas badań znalazłem najwięcej interesujących błędów.

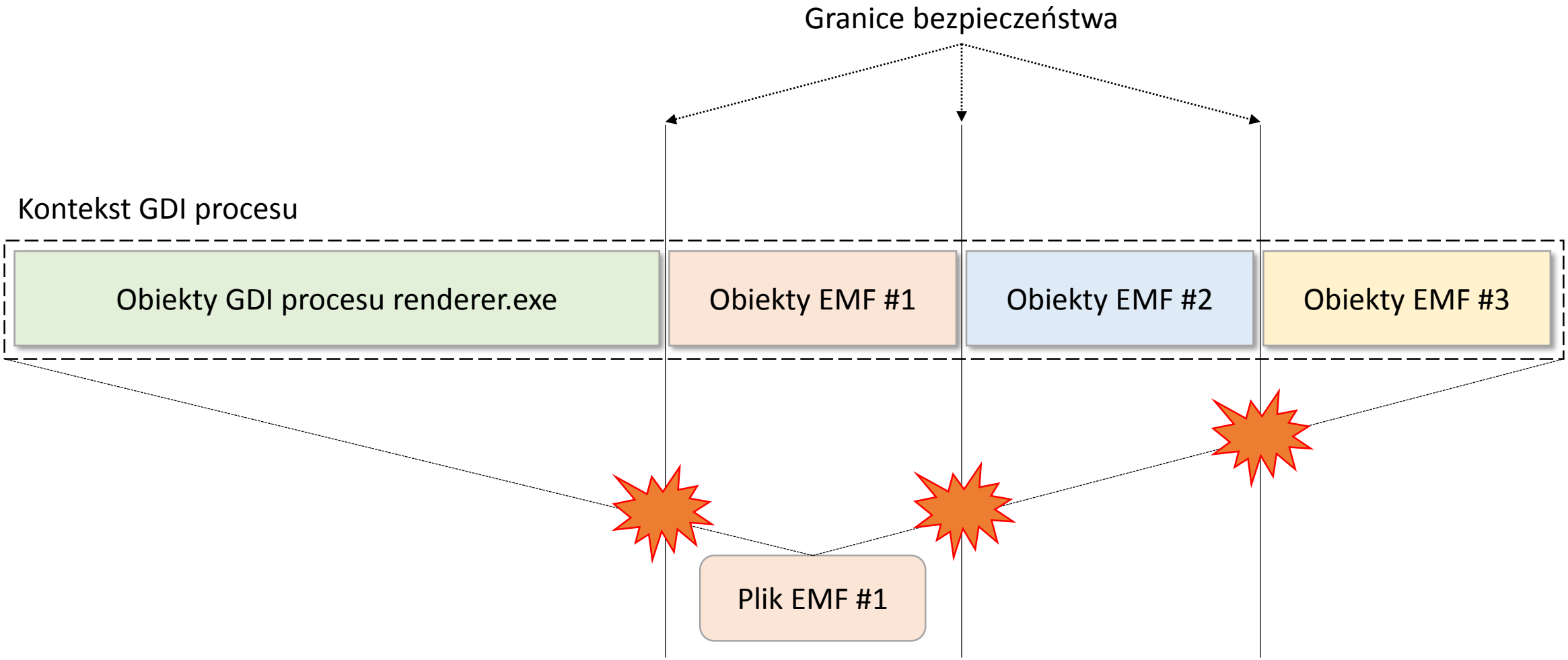
Scenariusz ataku

- We wszystkich przypadkach pliki metafile są przetwarzane w trybie użytkownika, w kontekście wyświetlającego procesu, w odpowiedniej bibliotece DLL.
  - GDI, GDI+ i MF3216 iterują po kolejnych rekordach wejściowych i tłumaczą je na odpowiednie wywołania GDI/GDI+.
- Błędy związane z obsługą pamięci poskutkują potencjalnie zdalnym wykonaniem kodu w tym kontekście.
- **Ważne:** pliki metafile operują bezpośrednio na kontekście GDI renderera.
  - Mogą tworzyć, usuwać, zmieniać i używać różnych obiektów GDI tworzonych w imieniu lokalnego procesu.
  - W teorii, każdy obrazek powinien mieć dostęp wyłącznie do własnych obiektów.
  - Z drugiej strony, potencjalne błedy w implementacji mogą pozwolić na dostęp do innych obiektów używanych przez ten sam program.
  - Ciekawy przypadek podniesienia uprawnień.

Scenariusz ataku



Scenariusz ataku



Możliwe rodzaje błędów

1. **Memory corruption**
   - Przepełnienia bufora itp. spowodowane błędną obsługą niektórych rekordów.
   - Potencjalnie eksploitowalne w dowolnej aplikacji.
   - Rezultat: zazwyczaj RCE.
2. **Ujawnienie pamięci**
   - Wyświetlanie niezainicjalizowanej lub wykraczającej poza ramy bufora pamięci jako piksele obrazka.
   - Eksploitowalne wyłącznie w kontekstach umożliwiających odczytanie kolorów pikseli z powrotem (np. przeglądarki internetowe, zdalne renderery).
   - Rezultat: ujawnienie informacji (kradzież tajnych danych, pokonanie ASLR etc.).
3. **Nieprawidłowa interakcja z systemem operacyjnym.**
   - Rezultat, eksploitacja = ???, zależy od natury konkretnego błędu.

Czas zacząć!

- Na początku roku zdecydowałem się przeaudytować dostępne implementacje EMF.
- Skutkiem jest 10 CVE u Microsoftu oraz 3 CVE u VMware (co odpowiada kilkudziesięciu rzeczywistym błędom).
- Przyjrzyjmy się przyczynom i eksploitacji najciekawszych z nich.
  - Przykłady opierają się na systemie Windows 7 32-bit, ale większość badań ma zastosowanie dla wszystkich platform z Windows 10 włącznie.

Audytowanie GDI

Zanim zaczniemy

- Aby zapoznać się z daną implementacją i sprawdzić, jakie błędy znaleziono w niej dotychczas, warto sprawdzić wyniki poprzednich badań.
- Zapytanie „wmf vulnerability” zwraca jeden wynik:

błąd SetAbortProc!

Błąd SetAbortProc w WMF (CVE-2005-4560)

- Odkryty 27 grudnia 2005 r. Naprawiony 5 stycznia 2006 r.
- Krytyczny błąd pozwalający na 100% skuteczne wykonanie kodu w momencie użycia GDI do wyświetlenia exploita (np. w Internet Explorerze).
- Nazwany „Windows Metafile vulnerability”, zwycięzca Pwnie Award 2007.
- Brak memory corruption, w użyciu wyłącznie udokumentowana funkcjonalność WMF.
- W czym więc tkwił problem?

Funkcja 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
);
```

wskaźnik na funkcję

... i jej odpowiednik w WMF

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.

W skrócie...

... format pozwalał na zlecenie następującego wywołania:

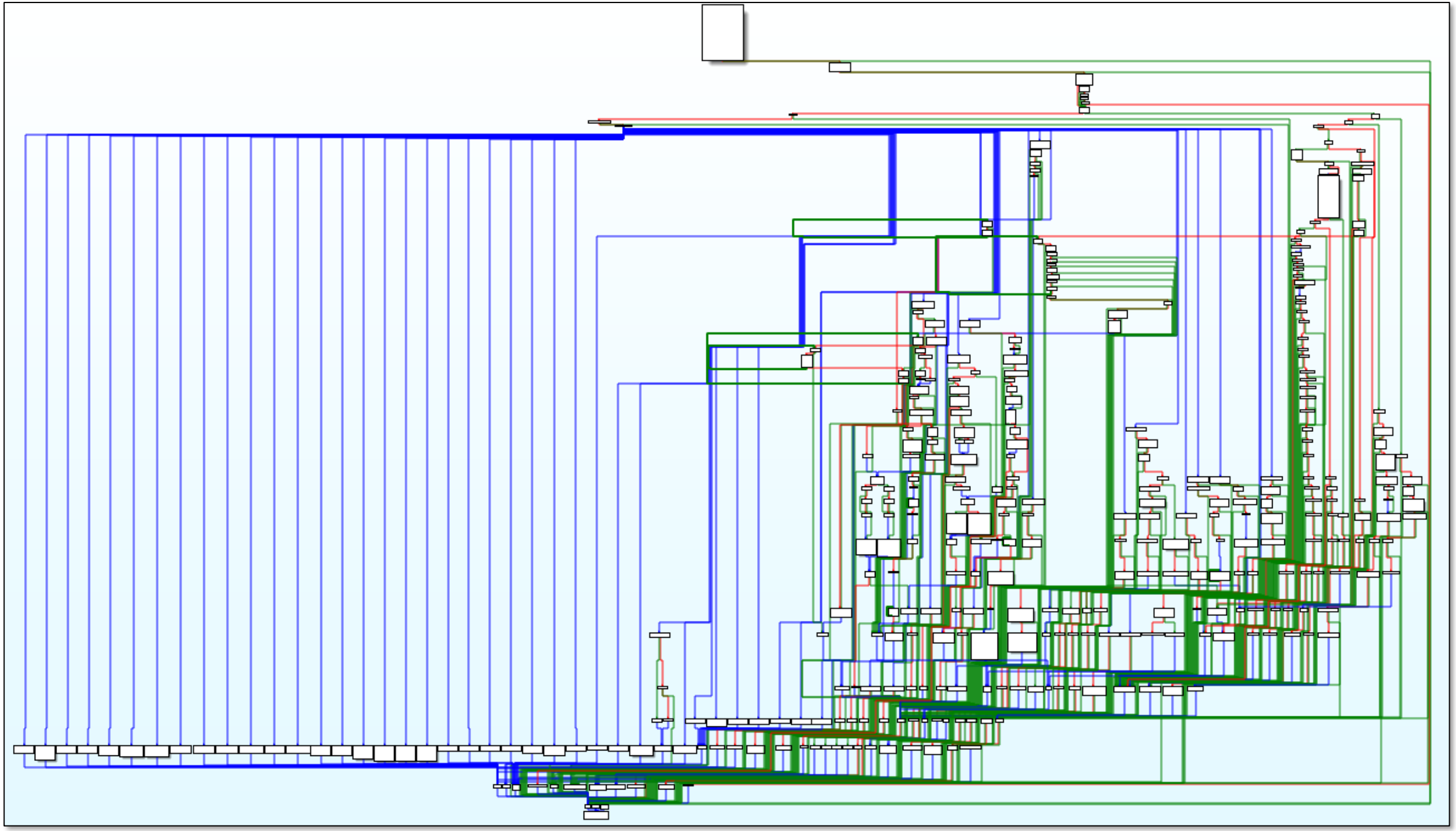
```
SetAbortProc(hdc, (ABORTPROC)"kontrolowane dane");
```

a następnie wywołanie tak zdefiniowanego wskaźnika na funkcję.

Z założenia wykonanie dowolnego kodu.

Zdobyta wiedza

1. Format może (nie)oficjalnie przekierowywać wywołania do ciekawych / niebezpiecznych funkcji, więc warto dokładnie przyglądać się działaniu każdej z nich i znaczeniu ich argumentów.
2. Cała obsługa rekordów WMF znajduje się w jednej, ogromnej funkcji `gdi32!PlayMetaFileRecord`.



Co z błędami w EMF?

- Zapytanie „emf vulnerability” zwraca więcej wyników.
- Najnowszy: „Yet Another Windows GDI Story” autorstwa Hosseina Lotfi (@hosselot).
  - Naprawiony w kwietniu 2015 r. w ramach MS15-035, CVE-2015-1645.
  - Przepełnienie bufora na stercie spowodowane przez niezweryfikowane założenie co do jednego z pól w rekordzie SETDIBITSTODEVICE.
  - W dużej mierze moja inspiracja do zajęcia się tym tematem.

Zdobyta wiedza

- Główna funkcja wykonująca rekordy EMF to `gdi32!PlayEnhMetaFileRecord`.
- Każdy rekord ma swoją własną klasę z dwoma metodami:
  - `::bCheckRecord()` – sprawdza integralność i poprawność danych rekordu.
    `::bPlay()` – wykonuje wywołania zdefiniowane w rekordzie.

Tablica GDI32 ::bCheckRecord

```
.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 *)
```

Tablica GDI32 ::bPlay

```
.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)
```

Dobry punkt wyjściowy.

CVE-2016-0168

Skutek:	Ujawnienie informacji o istnieniu plików
Rekordy:	EMR_CREATECOLORSPACE, EMR_CREATECOLORSPACEW
Eksploitowalny w:	Internet Explorer
CVE:	CVE-2016-0168
***google-security-research*:**	722
Naprawiony:	MS16-055, 10 maja 2016 r.

Mały błąd #1 w EMR_CREATECOLORSPACEW

- Jakość kodu może być łatwo oceniona po zaobserwowaniu wielu drobnych, ale oczywistych błędów.
- `MRCREATECOLORSPACEW::bCheckRecord()` sprawdza, czy rozmiar rekordu to co najmniej 0x50 bajtów:

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

- Następnie od razu odczytuje wartość pola .cbData pod przesunięciem 0x25C:

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

- Skutek: odczyt pamięci 0x20C bajtów poza buforem.

Mały błąd #2 w EMR_CREATECOLORSPACEW

- Dalej, pole `.cbData` spod nieprawidłowego przesunięcia 0x25C jest używane do sprawdzenia długości rekordu:

```
.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
```

- Powyższe tłumaczy się do:

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

Mały błąd #2 w EMR_CREATECOLORSPACEW

- Dwa problemy:
  1. Oczywisty integer overflow, pozwalający bardzo dużemu `.cbData` przejść weryfikację.
  2. Czemu długość rekordu miałaby być **mniejsza** niż długość zadeklarowanych w nim danych? Powinna być **większa lub równa**!
- W praktyce nie ma to znaczenia, bo owe dane i tak nie są dalej używane.

Mały błąd #3 w EMR_CREATECOLORSPACEW

- Brak sprawdzenia, czy bufor `.lcsFilename` w kontrolowanej strukturze `LOGCOLORSPACEW` jest domknięty bajtem zerowym.
  - Może prowadzić do niepoprawnego odczytu pamięci podczas obsługi łańcucha.
- Jasno widać, że kod przyjmuje po cichu wiele założeń co do danych wejściowych bez ich sprawdzania.
  - Na razie niegroźne, ale to i tak „dobry” znak.

Ujawnienie istnienia plików

- Wracając do działania rekordów EMR_CREATECOLORSPACE[W]: sprowadzają się one do wywołania funkcji `CreateColorSpace[W]` z kontrolowaną strukturą LOGCOLORSPACE:

```
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;
```

W środku CreateColorSpaceW

- Funkcja tworzy ścieżkę pliku z profilem kolorów przy użyciu wewnętrznego wywołania `gdi32!BuildIcmProfilePath`.
  - jeśli wejściowa nazwa jest względna, dołączana jest do predefiniowanej ścieżki systemowej.
  - w przeciwnym wypadku jest pozostawiana w oryginalnej formie.
- Wszystkie ścieżki są akceptowane, za wyjątkiem zaczynających się od dwóch znaków "/" lub "\":

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

W środku CreateColorSpaceW

- Prawdopodobnie ma to na celu zablokowanie zdalnych ścieżek UNC zaczynających się od "\\", np. \\192.168.1.13\C\Users\test\profile.icc.
- James Forshaw zauważył, że ta logika nie ma sensu, bo równoważnym prefiksem w Windowsie jest "\??\UNC\".
- Cała ścieżka omijająca sprawdzenie może wyglądać następująco:

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

CreateColorSpaceInternalW: ostatni krok

- Po sformatowaniu ścieżki, ale przed wywołaniem `NtGdiCreateColorSpace`, funkcja próbuje otworzyć plik i od razu go zamknąć, żeby sprawdzić, czy w ogóle istnieje:

```
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);
```

Konsekwencje

- W rezultacie możemy wywołać funkcję `CreateFileW()` na dowolnej zdefiniowanej przez nas ścieżce.
  - Jeśli wywołanie się powiedzie (plik istnieje), obiekt *color space* zostanie utworzony i funkcja zwróci sukces.
  - W przeciwnym wypadku obiekt GDI nie zostanie utworzony, a funkcja zwróci błąd.
- Wygląda na potencjalne ujawnienie informacji.
  - Jak podejść do wykorzystania tego zachowania np. w Internet Explorerze?

Intuicyjnie: ujawnienie zwracanej wartości

- Skoro status wywołania funkcji `CreateFileW()` determinuje, czy przetwarzanie rekordu się powiodło, może moglibyśmy to ujawnić?
  - Pierwszy pomysł: użyć `EMR_CREATECOLORSPACE` jako pierwszego rekordu, za którym następuje operacja rysowania.
  - Jeśli rysowanie nigdy nie jest wykonane (co można sprawdzić np. przy użyciu tagu <canvas>), wywołanie funkcji nie powiodło się.

Intuicyjnie: ujawnienie zwracanej wartości

- Niestety niemożliwe.
- Funkcja `gdi32!_bInternalPlayEMF` (wywoływana przez `PlayEnhMetaFile`) nie kończy przetwarzania obrazka w momencie problemu z jednym rekordem.
  - Flaga „sukces” jest ustawiana na FAŁSZ, a pętla interpretera kontynuuje działanie.
- Po przetworzeniu wszystkich rekordów flaga jest zwracana.

Ujawnienie końcowego statusu dla pliku?

- Również niemożliwe.
- Wartość zwracana funkcji `PlayEnhMetaFile` jest ignorowana przez Internet Explorer w `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]
```

Pozostałe opcje

- Innym indykatorem może być utworzenie obiektu *color space* przez wywołanie systemowe `NtGdiCreateColorSpace`.
- Ujawnienie tego bezpośrednio jest trudne (o ile w ogóle możliwe), ale być może istnieje jakiś *side channel*?

Użycie limitu obiektów GDI

- Każdy proces w Windowsie jest domyślnie ograniczony do 10,000 obiektów GDI.
  - Liczba ta może być zmieniona w rejestrze, ale nie jest w przypadku Internet Explorera.
- Jeśli użyjemy 10,000 rekordów `EMR_CREATECOLORSPACEW` ze ścieżką pliku do sprawdzenia, to:
  - Jeśli plik istnieje, wygenerujemy 10,000 obiektów *color space*, osiągając maksymalną granicę.
  - Jeśli nie istnieje, żaden obiekt nie zostanie utworzony.
- W tym momencie albo dobiliśmy do limitu, albo nie. Kiedy spróbujemy utworzyć pędzel (kolejny obiekt) i użyć go do rysowania, to:
  - Jeśli plik istnieje, utworzenie pędzla nie powiedzie się, i zostanie użyty domyślny.
  - Jeśli nie istnieje, pędzel zostanie utworzony i wykorzystany do malowania.

Limit obiektów GDI jako wyrocznia – ilustracja

DEMO

Skutki podatności

- Ujawnienie informacji o istnieniu wybranych plików, przydatne do:
  - Rozpoznawania oprogramowania (i jego wersji) zainstalowanego na komputerze ofiary.
  - Śledzenie użytkowników (poprzez tworzenie profili bazujących na wykrytych plikach).
  - Śledzenie momentu otwarcia dokumentów offline (np. każdorazowe otwarcie dokumentu Worda wysyłające sygnał do zdalnego serwera przy użyciu ścieżki UNC).
  - Skanowanie dostępnych zasobów sieciowych dostępnych dla lokalnego użytkownika.

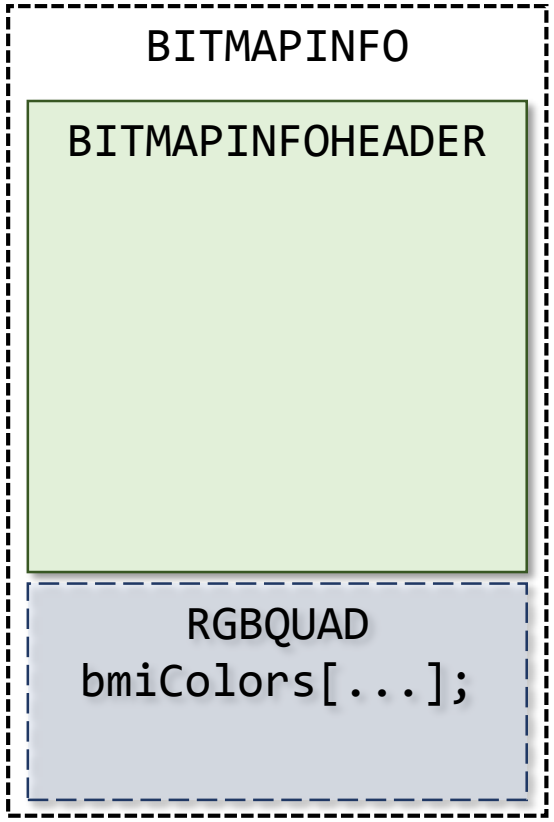
CVE-2016-3216

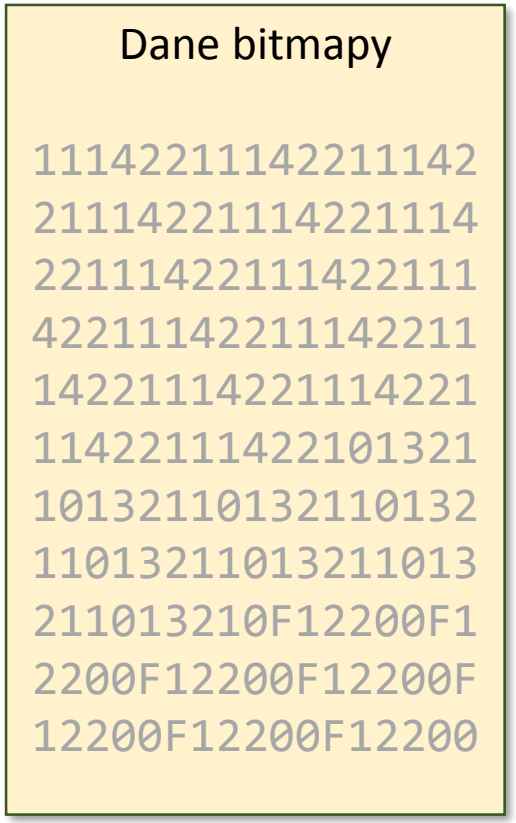
Skutek:	Ujawnienie pamięci
Rekordy:	Wiele rekordów (10)
Eksploitowalny w:	Internet Explorer
CVE:	CVE-2016-3216
***google-security-research*:**	757
Naprawiony:	MS16-074, 14 czerwca 2016 r.

Device Independent Bitmaps (DIBs)

W Windows GDI, bitmapy są zazwyczaj przechowywane w formie tzw. „DIB”:

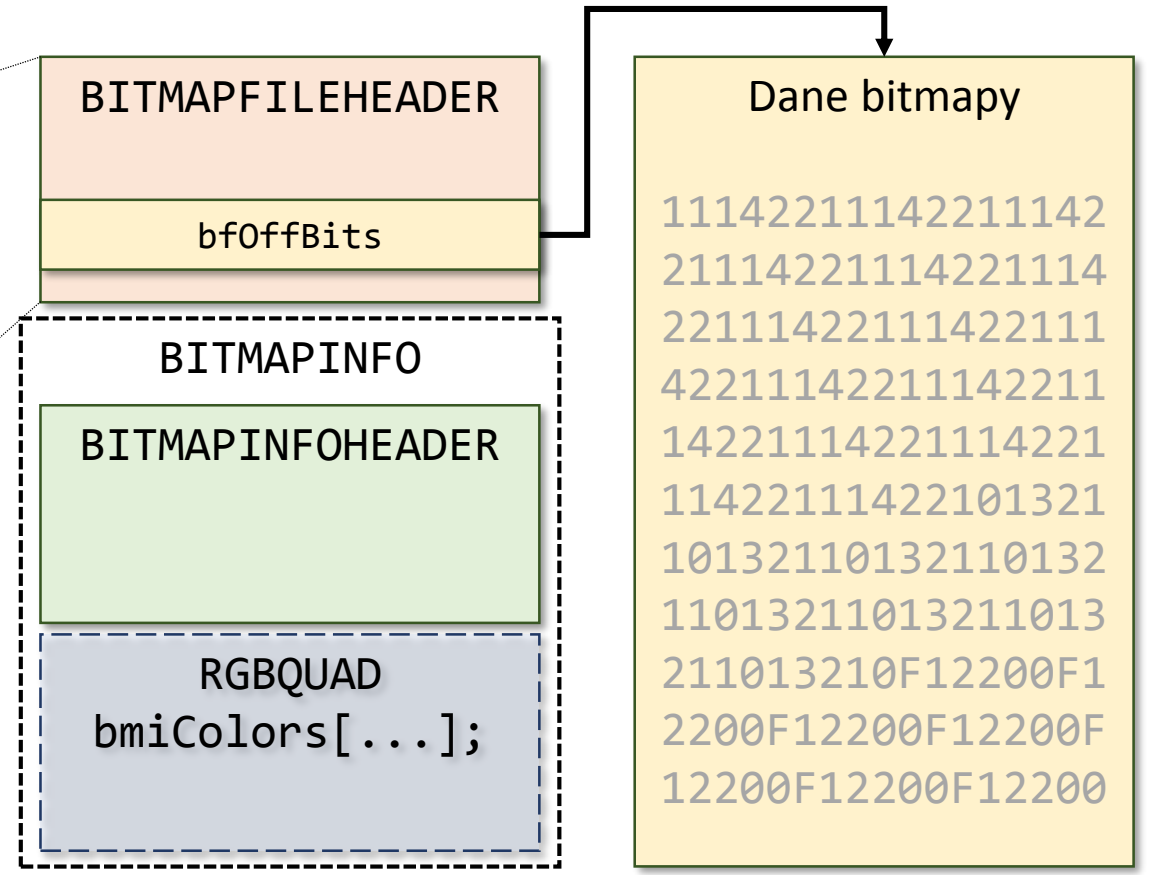
- Krótki nagłówek zawierający podstawowe metadane na temat obrazka, po którym opcjonalnie następuje paleta kolorów.
- Właściwe dane bitmapy.





Pliki .BMP to również DIB

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



BITMAPINFOHEADER, trywialny nagłówek

```
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;
```

- Krótka i prosta struktura.
- 40 bajtów długości (w typowej formie).
- Tylko 8 istotnych pól.

Czy aby na pewno trywialny?

- `biSize` musi zostać sprawdzone.
- `biWidth`, `biHeight`, `biPlanes`, `biBitCount` mogą spowodować przepełnienie typu całkowitego (często są przez siebie mnożone).
- `biHeight` może być ujemne, co wskazuje na bitmapę zapisaną „do góry nogami”.
- `biPlanes` musi być równe 1.
- `biBitCount` musi należeć do zbioru {1, 2, 4, 8, 16, 24, 32}.
  - Jeśli `biBitCount` < 16, może zostać użyta paleta kolorów.
  - Na rozmiar palety kolorów ma również wpływ pole `biClrUsed`.

Czy aby na pewno?

- `biCompression` musi być równe `BI_RGB`, `BI_RLE8`, `BI_RLE4`, `BI_BITFIELDS`, ...
  - Każdy algorytm kompresji musi być poprawnie obsłużony.
- `biSizeImage` musi odpowiadać właściwemu rozmiarowi obrazka.
- Paleta kolorów musi być wystarczająco długa, by pomieścić wszystkie potrzebne elementy.
- Danych obrazu musi być wystarczająco dużo, by opisać wszystkie piksele.
- Zakodowane piksele muszą odpowiadać wartościom z nagłówków (nie wykraczać poza rozmiar palety itp.).

Wiele potencjalnych problemów

1. Drzewo decyzji poprawnej obsługi DIB jest bardzo skomplikowane.
2. Istnieje wiele przypadków brzegowych, o których trzeba pamiętać, by uniknąć błędów.
3. Konieczna jest konsekwencja w sposobie interpretowania wartości w nagłówkach.

Funkcje GDI operujące na DIB (bezpośrednio)

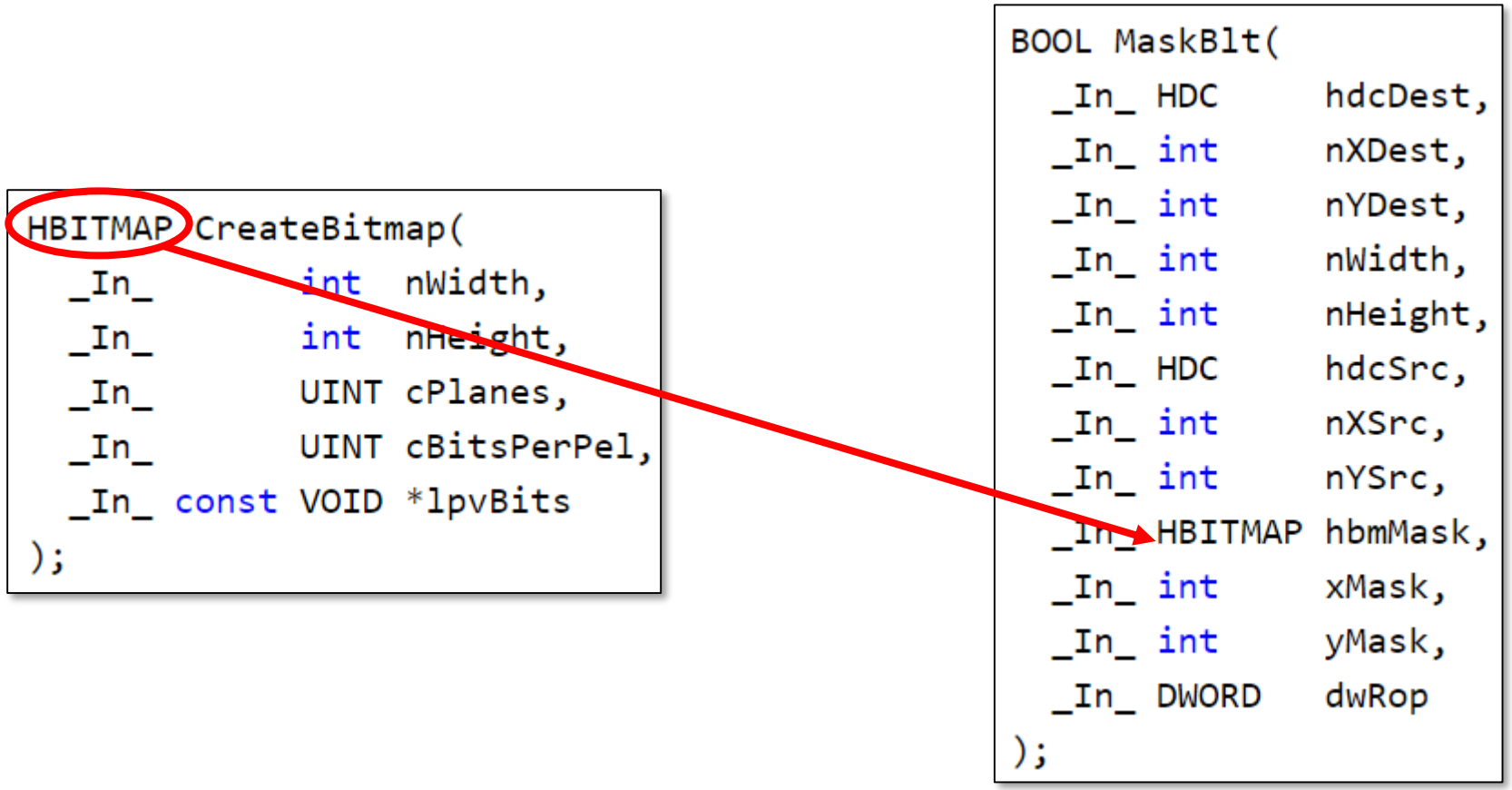
```
int StretchDIBits(
  _In_       HDC        hdc,
  _In_       int        XDest,
  _In_       int        YDest,
  _In_       int        nDestWidth,
  _In_       int        nDestHeight,
  _In_       int        XSrc,
  _In_       int        YSrc,
  _In_       int        nSrcWidth,
  _In_       int  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

wskaźniki na dane bitmapy

wskaźniki na nagłówki DIB

```
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
);
```

Funkcje GDI operujące na DIB (pośrednio)



Odpowiedzialność za walidację danych

- We wszystkich przypadkach, odpowiedzialność za poprawność nagłówków i danych spoczywa na kodzie wywołującym funkcje API.
- Przekazywanie bitmap w pełni kontrolowanych przez użytkownika jest problematyczne, gdyż wymaga obsługi wszystkich wymienionych wcześniej przypadków brzegowych.
- Zgadnijmy co? EMF wspiera znaczną liczbę rekordów zawierających osadzone struktury DIB.

Rekordy EMF zawierające DIB

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

Wspólny schemat

- Dwie pary `(offset, size)` dla nagłówka oraz danych bitmapy:

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.

Rodzaje niezbędnej weryfikacji

- W każdej funkcji obsługującej rekord z DIB powinny się znaleźć cztery warunki:
  1. `cbBmiSrc` jest odpowiednio duży dla całego nagłówka.
  2. `(offBmiSrc, offBmiSrc + cbBmiSrc)` znajduje się wewnątrz rekordu.
  3. `cbBitsSrc` jest odpowiednio duży dla kompletnych danych bitmapy.
  4. `(offBitsSrc, offBitsSrc + cbBitsSrc)` znajduje się wewnątrz rekordu.

Wiele brakujących warunków

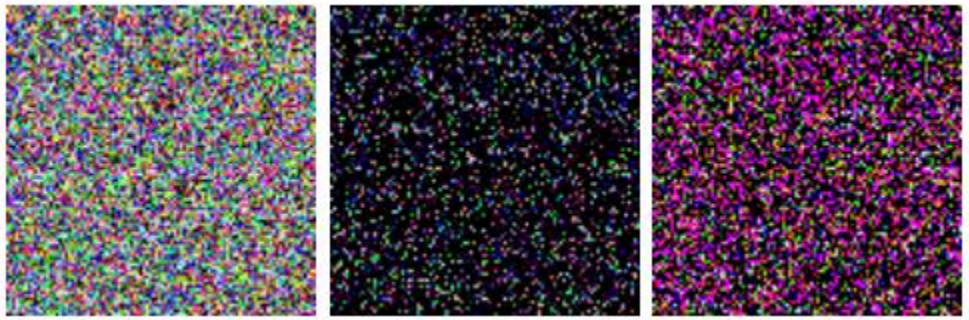
Metody obsługi rekordów	Brakujące warunki
MRALPHABLEND::bPlay MRBITBLT::bPlay MRMASKBLT::bPlay MRPLGBLT::bPlay MRSTRETCHBLT::bPlay MRTRANSPARENTBLT::bPlay	#1, #2
MRSETDIBITSTODEVICE::bPlay	#3
MRSTRETCHDIBITS::bPlay	#1, #3
MRSTRETCHDIBITS::bPlay MRCREATEMONOBRUSH::bPlay MREXTCREATEPEN::bPlay	#1, #2, #3, #4

Konsekwencje

- W związku z brakującą walidacją, niektóre części regionów opisujących obrazek mogą zostać załadowane z nieprawidłowych części sterty.
  - Nagłówek DIB
  - Paleta kolorów
  - Dane bitmapy
- Niezainicjalizowane lub wykraczające poza bufor dane sterty mogą zostać wyświetlone w postaci pikseli na ekranie.

Proof of concept

- Aby zweryfikować odkrycie, stworzyłem prosty plik EMF z rekordem `EMR_STRETCHBLT`, który zawierał 8-bitową bitmapę z paletą kolorów wykraczającą poza obręb pliku.
- Rezultat: „śmieciowe” dane wyświetlane na ekranie w postaci kolorów R, G, B.
- Ten sam obrazek załadowany trzy razy pod rząd w Internet Explorerze:



- Ujawnione dane mogą zostać odczytane z powrotem przy użyciu HTML5, w celu zdradzenia adresów bazowych obrazów wykonywalnych bądź zdobycia dostępu do innych wrażliwych danych.

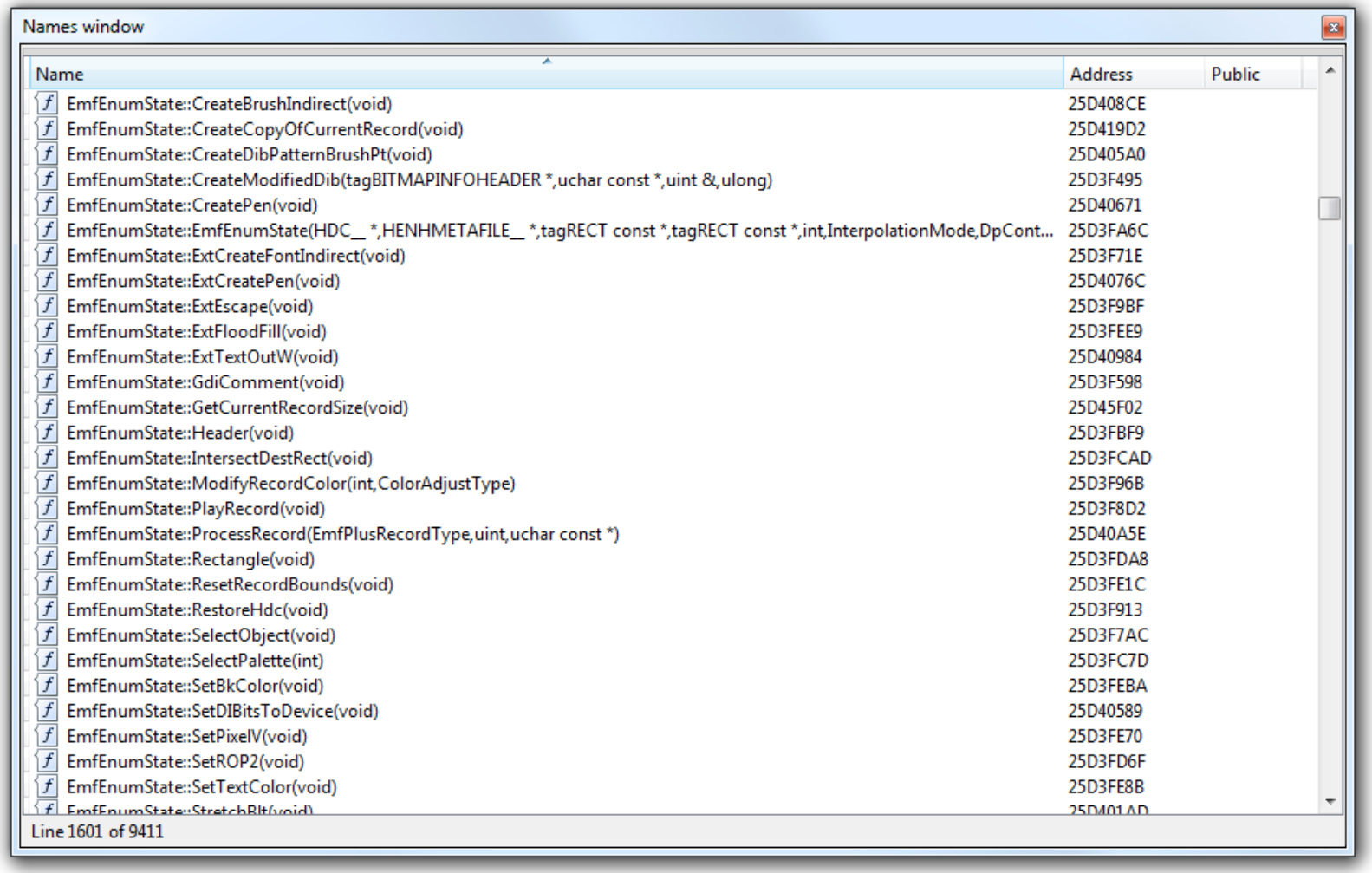
DEMO

Audytowanie GDI+

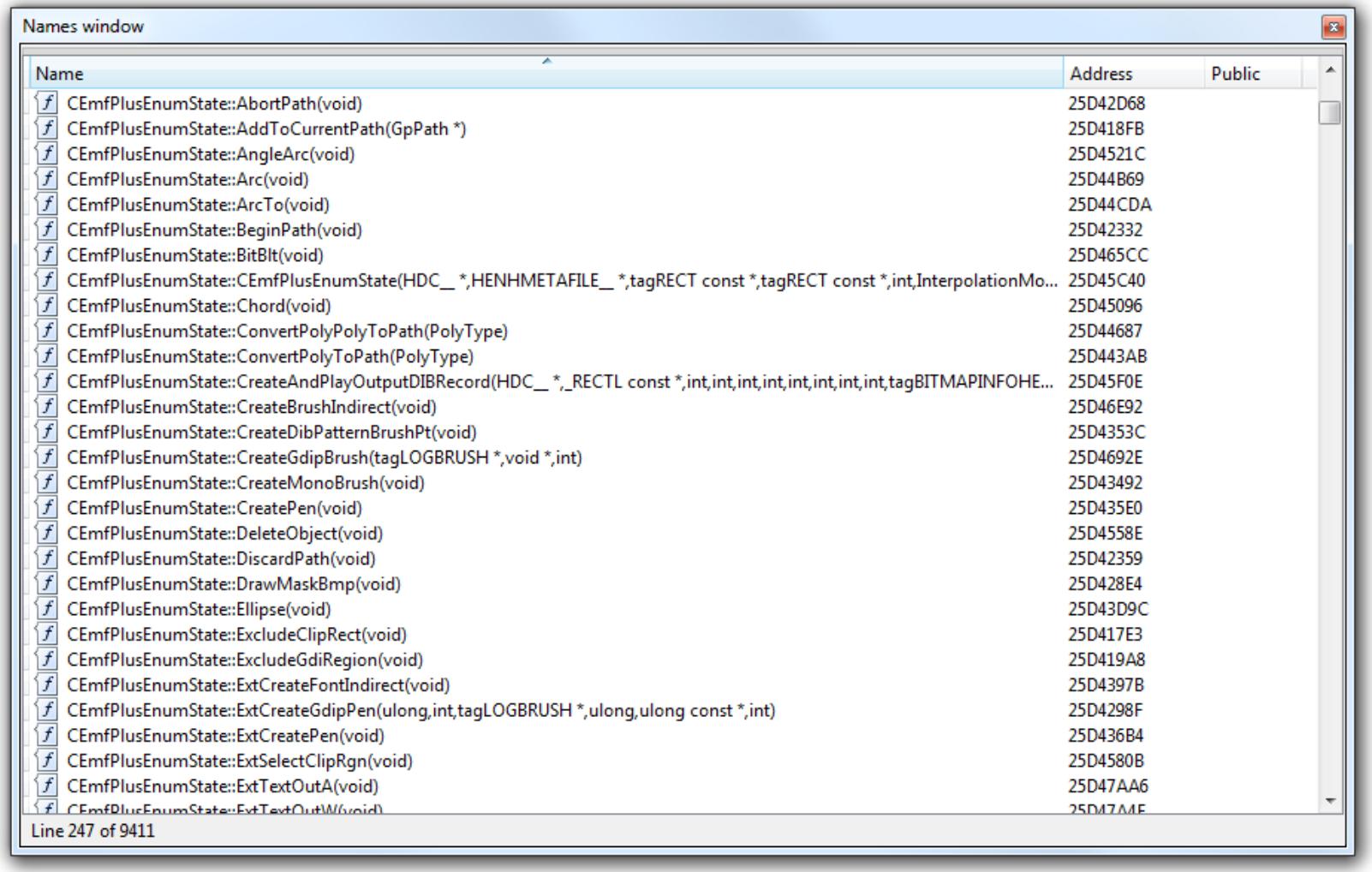
GDI+ jako cel ataku

- GDI+ wspiera zarówno EMF jak i EMF+.
  - Większość kodu jest niezależna, ale w niektórych miejscach znajdziemy bezpośrednie odwołania do GDI.
  - W związku z tym niektóre błędy w GDI mogą również dotyczyć GDI+.
- Najważniejszym klientem GDI+ jest prawdopodobnie pakiet Microsoft Office.
- Podobnie jak wcześniej, przystąpmy do audytu wszystkich funkcji obsługujących poszczególne rekordy EMF.

Powierzchnia ataku łatwa do znalezienia



Powierzchnia ataku łatwa do znalezienia



Przeanalizujmy konkretne błędy.

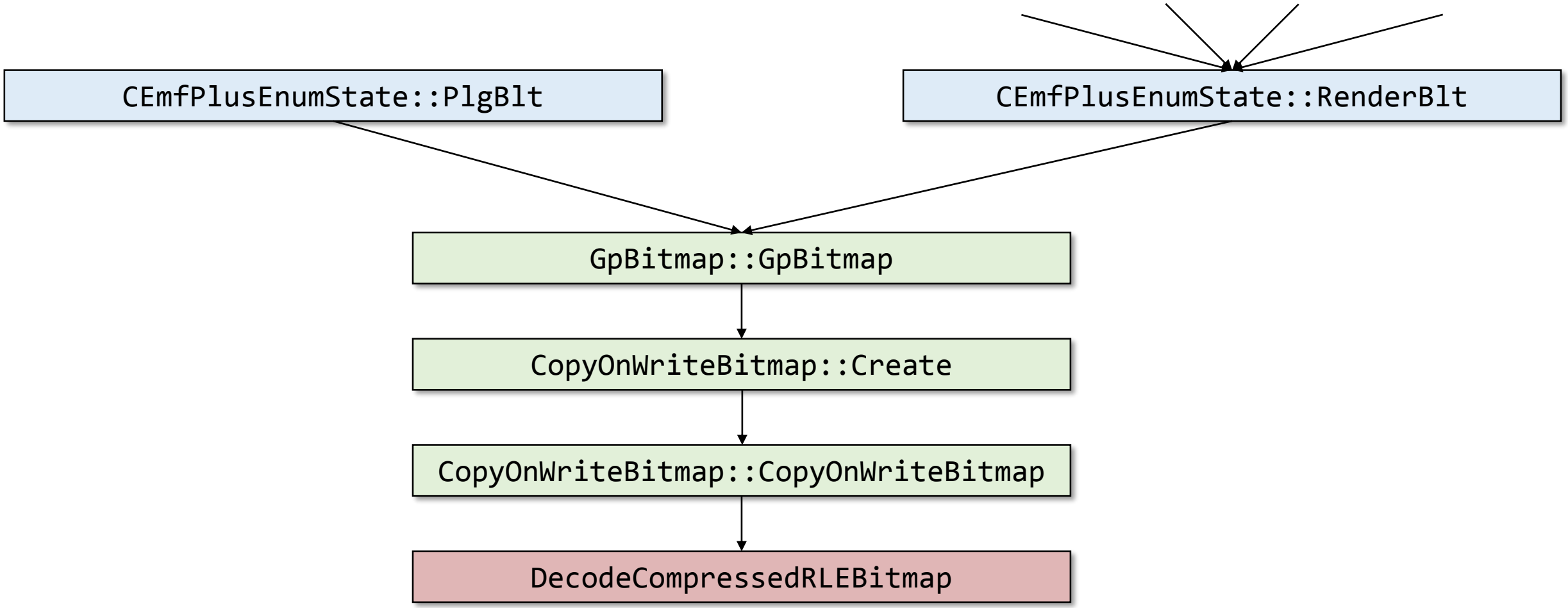
CVE-2016-3301

Skutek:	Write-what-where
Rekordy:	Wszystkie rekordy operujące na DIB
Eksploitowalny w:	Microsoft Office
CVE:	CVE-2016-3301
***google-security-research*:**	824
Naprawiony:	MS16-097, 9 sierpnia 2016 r.

Bitmapy skompresowane RLE w EMF

- Jak wcześniej wspomniałem, wiele rekordów EMF zawiera w sobie DIB.
- Bitmapy mogą być skompresowane przy użyciu kilku prostych algorytmów, takich jak 4- i 8-bitowe *Run Length Encoding*.
  - Oznaczone przez odpowiednią wartość pola `biCompression`.
- Podczas przekopywania się przez kod handlerów odkryłem, że 8-bitowe RLE jest wspierane przez GDI+.
- Dekompresja RLE była historycznie źródłem wielu błędów bezpieczeństwa.

Dotarcie do kodu



W `DecodeCompressedRLEBitmap()`

- Obliczane są dwie wartości:

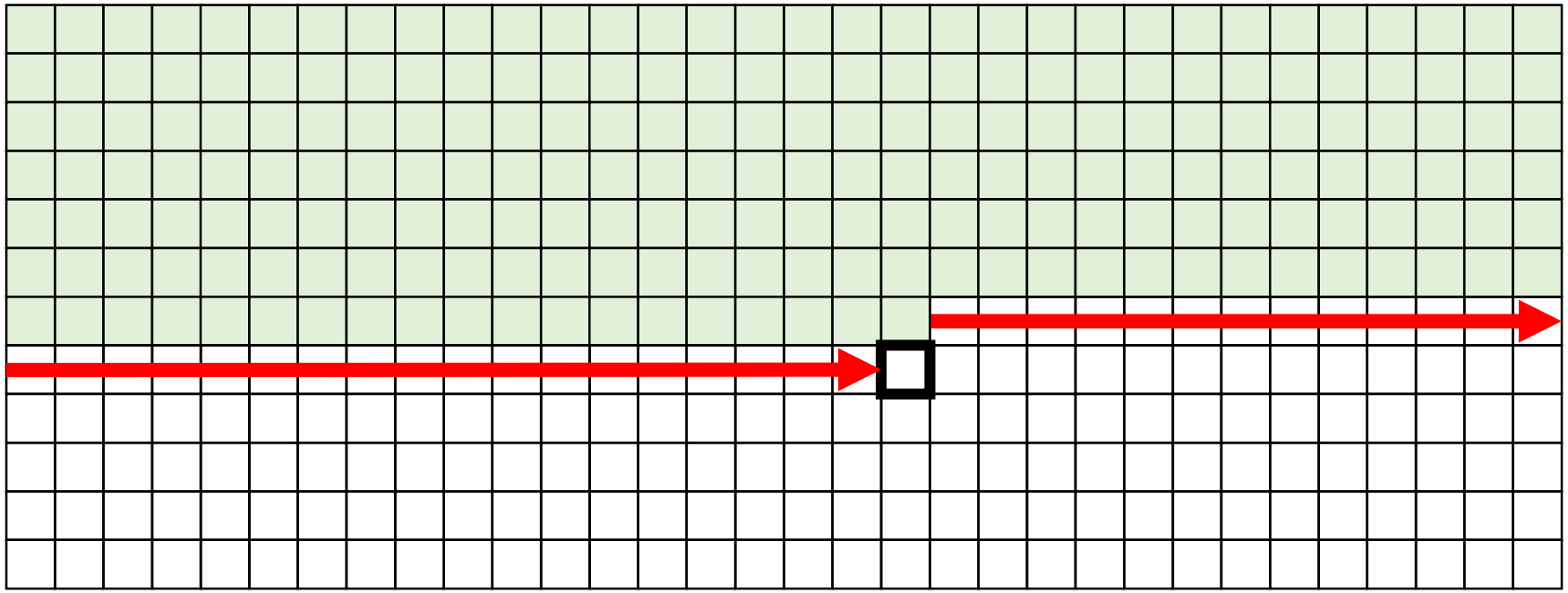
```
columns = abs(biHeight)
```

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

- Bufor wyjściowy o rozmiarze `columns * bytes_per_row` jest alokowany ze sterty.
  - Duży stopień kontroli nad rozmiarem obszaru pamięci.
- Rozpoczyna się interpretacja i „wykonanie” programu RLE.

Opkod „End of Line”

- Przesuwa wskaźnik bufora wyjściowego do następnej linii.



Opkod „End of Line”

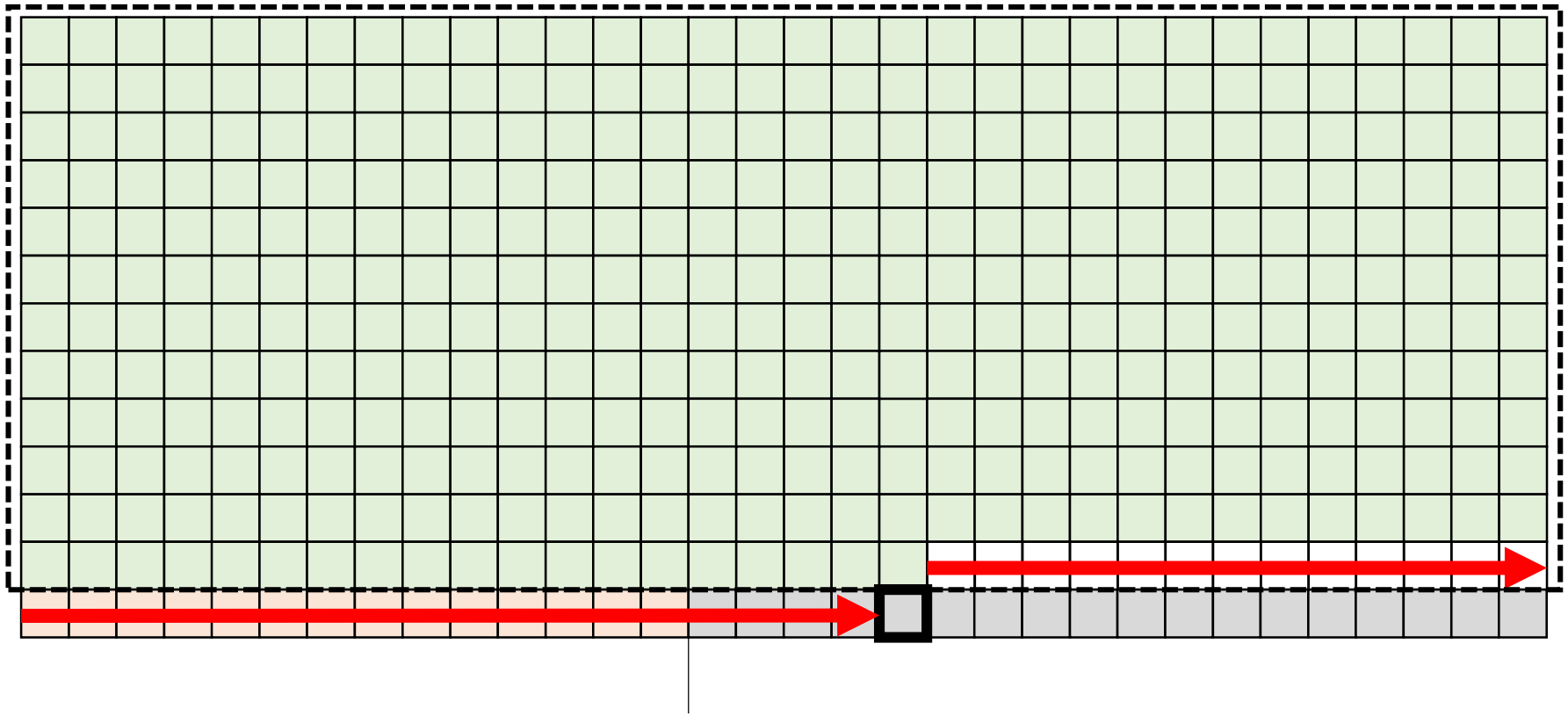
- W GDI+ zaimplementowane w następujący sposób:

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

- Kod sprawdza, czy wskaźnik nia znalazł się poza dozwolonymi ramami.
- Faktycznie działa poprawnie na 64-bitowych platformach, ale czy powyższy warunek jest na pewno wystarczający?

Niewystarczająca walidacja

Bufor wyjściowy



Koniec przestrzeni adresowej procesu

`0xffffffff`

Trudna arytmetyka wskaźników

- Dla bardzo szerokich bitmap, odległość pomiędzy aktualnym wskaźnikiem wyjściowym a końcem przestrzeni adresowej może być mniejsza, niż rozmiar wiersza.
- Wyrażenie:

```
out_ptr += bytes_per_row;
```

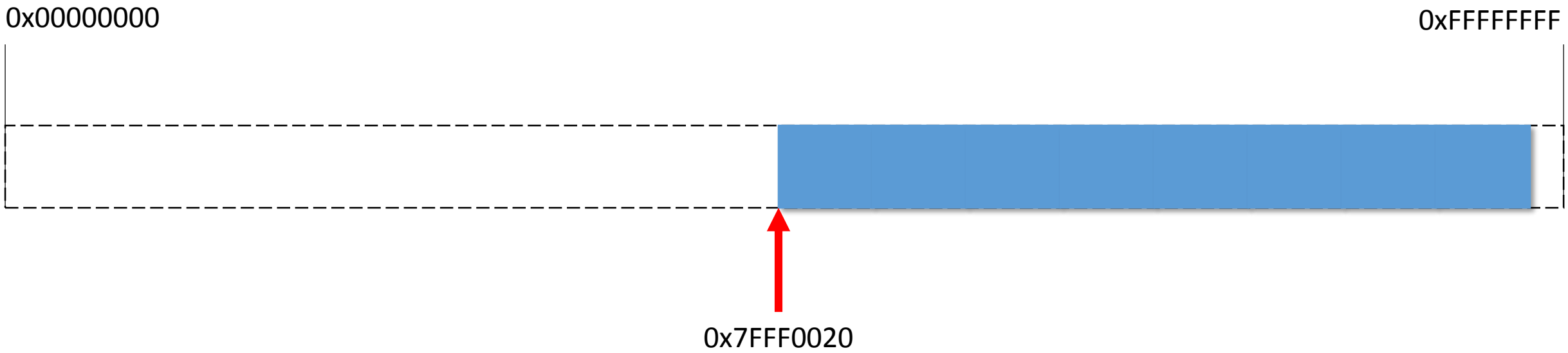
może spowodować przepełnienie typu wskaźnika, co z kolei zneutralizuje sprawdzenie warunku, który po nim następuje.

- W rezultacie możemy ustawić wskaźnik wyjściowy na w znacznym stopniu kontrolowaną wartość bezwzględną.

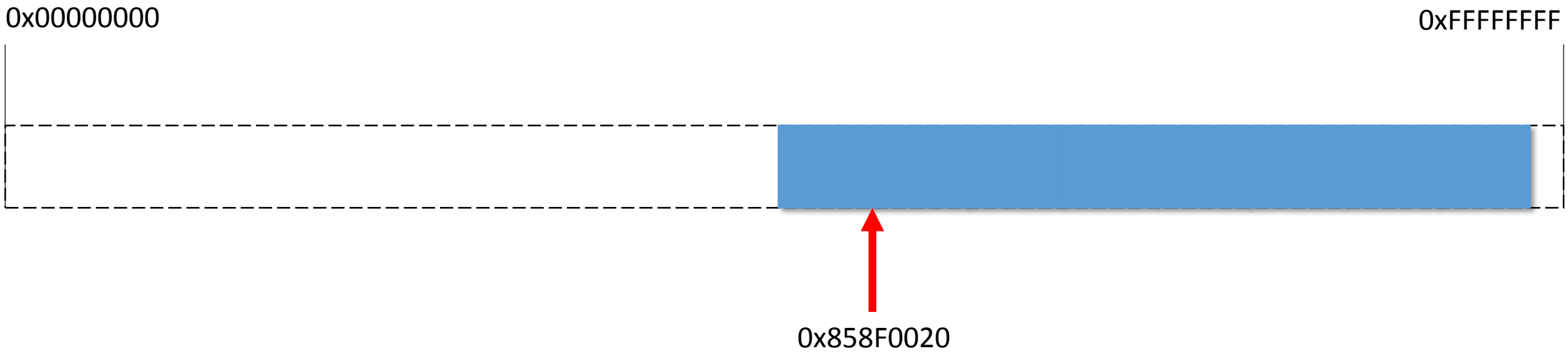
Przykład

- `biWidth` = **0x05900000**
- `biHeight` = **0x00000017**
- `biPlanes` = **0x0001**
- `biBitCount` = **0x0008**
- W rezultacie `columns` = **0x17** i `bytes_per_row` = **0x590000**.
- Całkowity rozmiar bufora = **0x7FF00000** (prawie 2 GB).
- Przykładowy adres początku alokacji: **0x7FFFF0020**, koniec: **0xFFEF0020**.

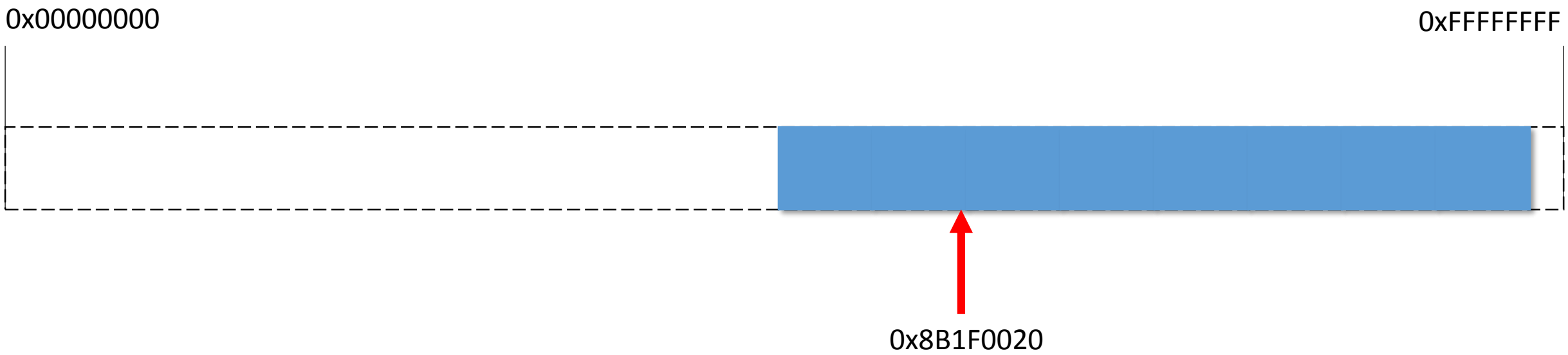
Układ przestrzeni adresowej



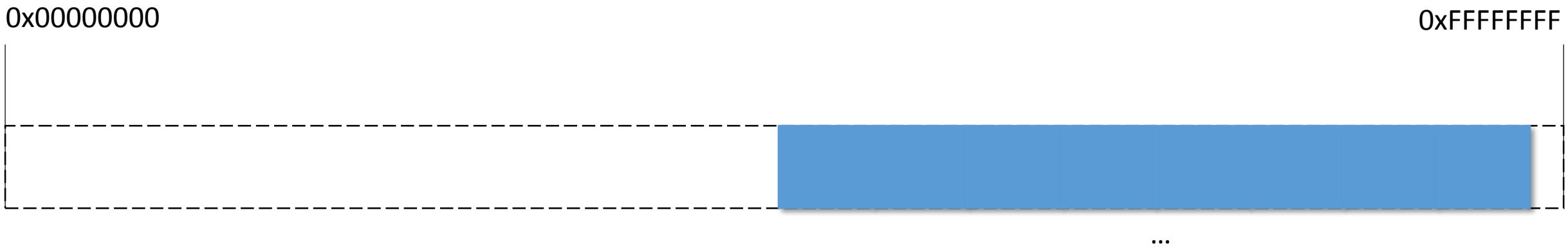
Układ przestrzeni adresowej (EOL #1)



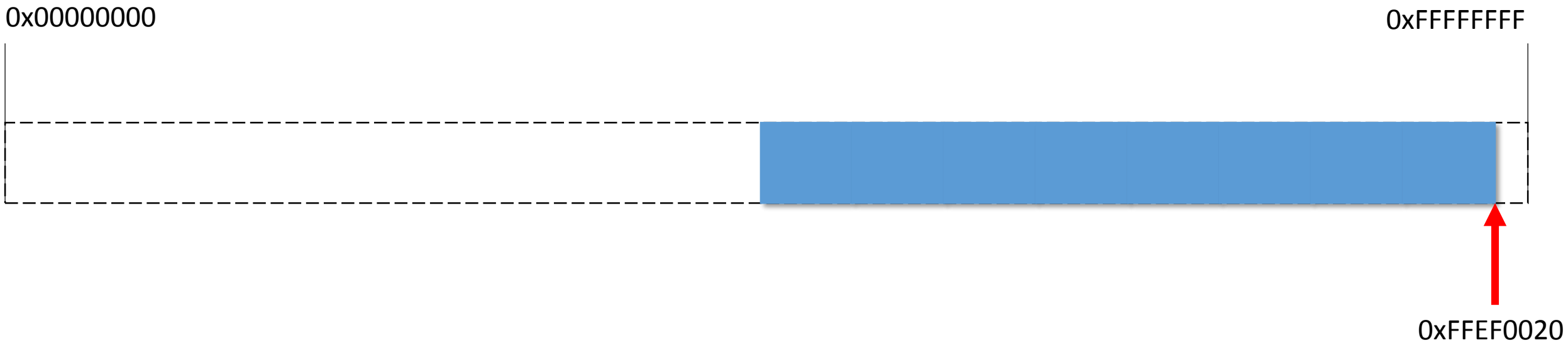
Układ przestrzeni adresowej (EOL #2)



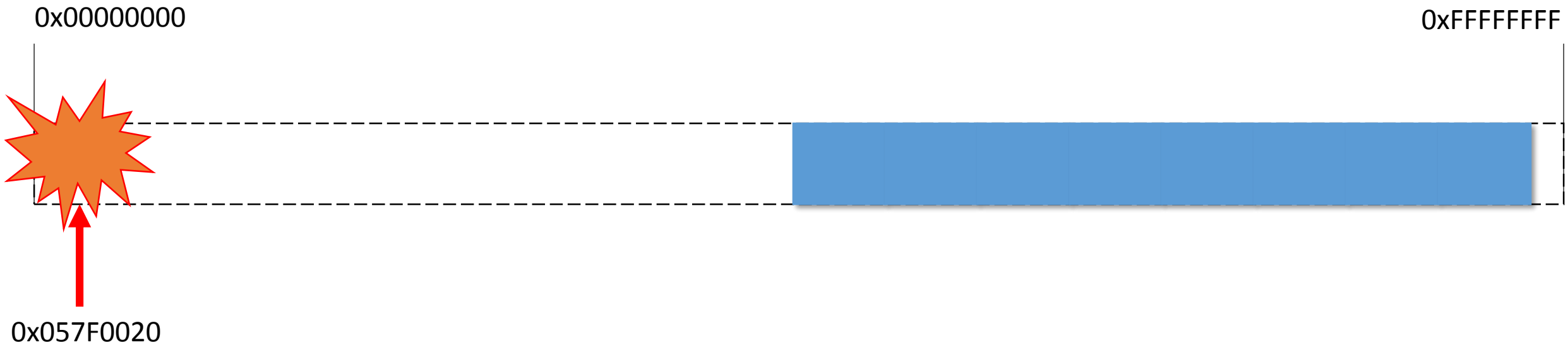
Układ przestrzeni adresowej (EOL #3-22)



Układ przestrzeni adresowej (EOL #23)



Układ przestrzeni adresowej (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
```

…

Podsumowanie

- Wymagania: 32-bitowy proces z włączoną obsługą PAE.
  - Pełna przestrzeń adresowa o rozmiarze 4GB musi być dostępna dla programu.
- Wynik: tzw. *write-what-where condition*, z bardzo wysokim stopniem kontroli „where”.
  - Prawie dowolny adres, z tym zastrzeżeniem, że poniżej adresu oryginalnego bufora wyjściowego.
- Stabilna eksploitacja jest bardzo zależna od stanu przestrzeni adresowej procesu w momencie ładowania spreparowanego obrazka.

Błędy ujawnienia pamięci w GDI+

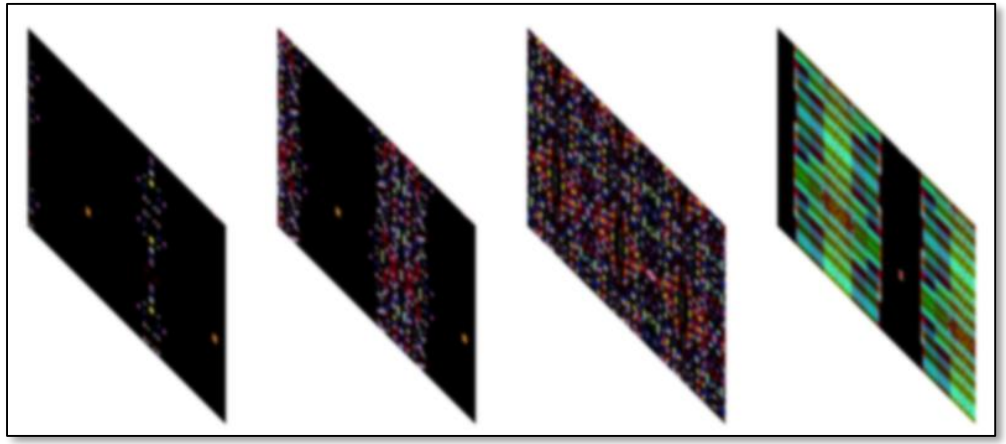
Skutek:	Ujawnienie pamięci sterty
Rekordy:	Wszystkie rekordy zawierające DIB
Eksploitowalne w:	Microsoft Office Online
CVE:	CVE-2016-3262, CVE-2016-3263
***google-security-research*:**	825, 829
Naprawione:	MS16-120, 11 października 2016 r.

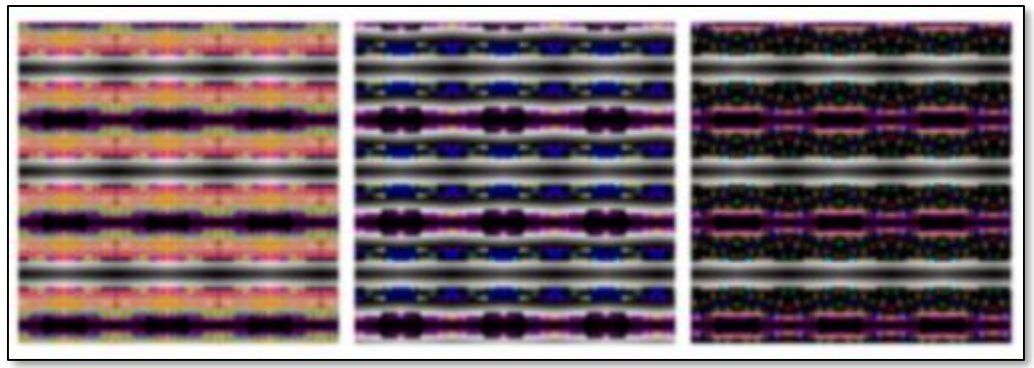
GDI+ vs DIB

- Podobnie jak w GDI, w GDI+ nie udało się uniknąć pewnych błędów ujawnienia informacji związanych z obsługą osadzonych bitmap.
- W szczególności:
    1. Jeśli dane skompresowanej bitmapy zaczynają się markerem „End of bitmap”, to cały obszar wyjściowy obrazu pozostaje niezainicjalizowany (a więc zawiera dane poprzednich alokacji na stercie).
    2. Brak weryfikacji, czy cała paleta kolorów bitmapy faktycznie zawiera się w obrębie danego rekordu EMF.

Naocznie widoczne błędy

- Po załadowaniu specjalnie spreparowanych obrazków do Worda, od razu rzuca się w oczy, że niezainicjalizowane dane są wyświetlane jako piksele.





Zdalna eksploitowalność?

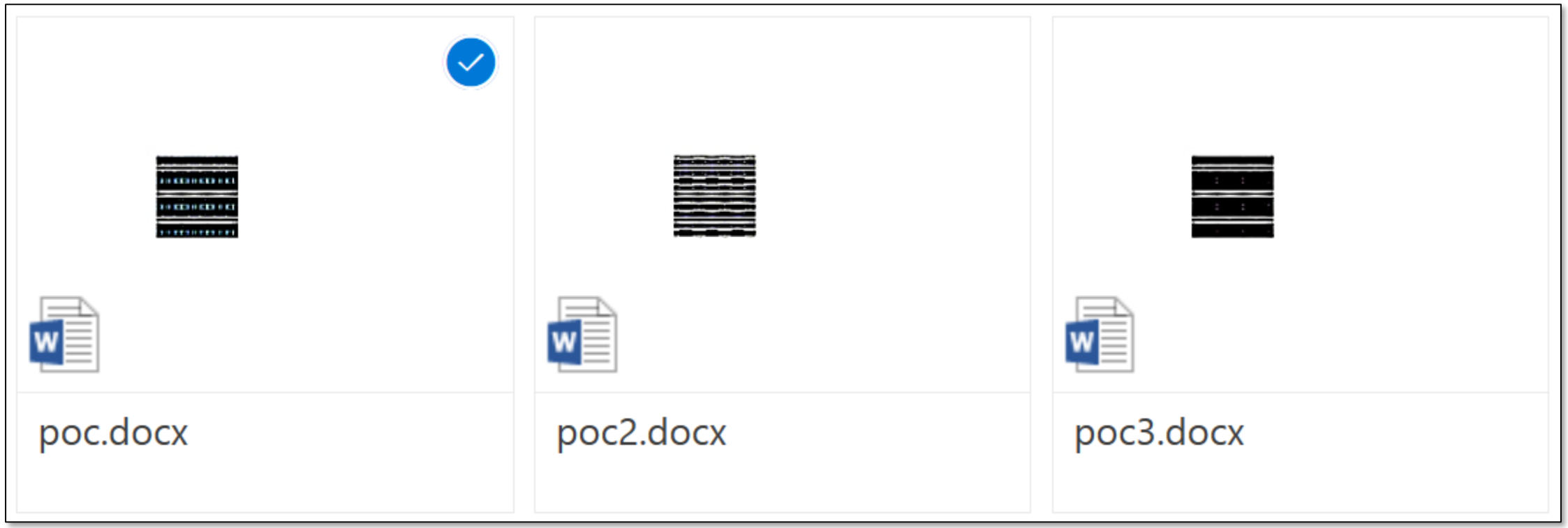
- Samo wyświetlanie niezainicjalizowanych danych nie jest poważnym problemem, jeśli nie można ich następnie w jakiś sposób odczytać.
- Jedynym oczywistym celem są programy z pakietu Microsoft Office, w których nie możemy wchodzić w żadną interakcję z wyświetlanym obrazem.
- Pomimo to błędy zostały zgłoszone do Microsoftu, aby sami poddali ocenie wagę podatności.
- MSRC oznaczył zgłoszenia jako „vNext” (nie załatane w biuletynie, potencjalnie naprawione w kolejnych wersjach biblioteki).

Ocena stopnia zagrożenia

- Początkowo zgodziłem się z tą decyzją, która była zbieżna z moją wiedzą.
  - Błędy #825 i #829 w *trackerze* Project Zero zostały otwarte kolejno 26 lipca i 9 sierpnia.
- Na początku sierpnia Ivan Fratric wspomniał, że błędy w GDI+/EMF mogą być również eksploitowalne zdalnie, w Office Online.
  - Nie miałem wcześniej pojęcia, że taki pakiet w ogóle istnieje.
  - Szczególnie interesujące w kontekście błędów ujawnienia pamięci w GDI+, które w innych warunkach nie mogłyby zostać wykorzystane.
  - Co prawda nie można osadzać własnych obrazków EMF w nowych dokumentach, ale te znajdujące się w importowanych plikach .docx są poprawnie obsługiwane.

Office Online

- Postanowiłem zweryfikować to kilka tygodni później, i…



Office Online

- Przy każdym wysłaniu/otwarciu dokumentu testowy obrazek był wyświetlany w inny sposób.
- Wniosek: zdalne ujawnienie pamięci procesu renderera na serwerach Microsoftu.
- Od razu przesłaliśmy nowe informacje do MSRC do ponownego rozważenia.
- W odpowiedzi dowiedzieliśmy się, że scenariusz z Office Online faktycznie nie został wcześniej przewidziany, a błędy w takim wypadku zostaną naprawione.
  - Niecały miesiąc temu, w październikowym Patch Tuesday.

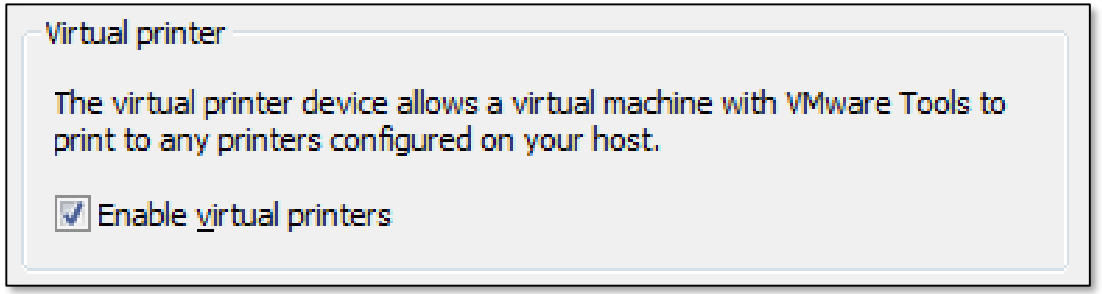
Hakowanie VMware Workstation

EMF a drukowanie (*print spooling*)

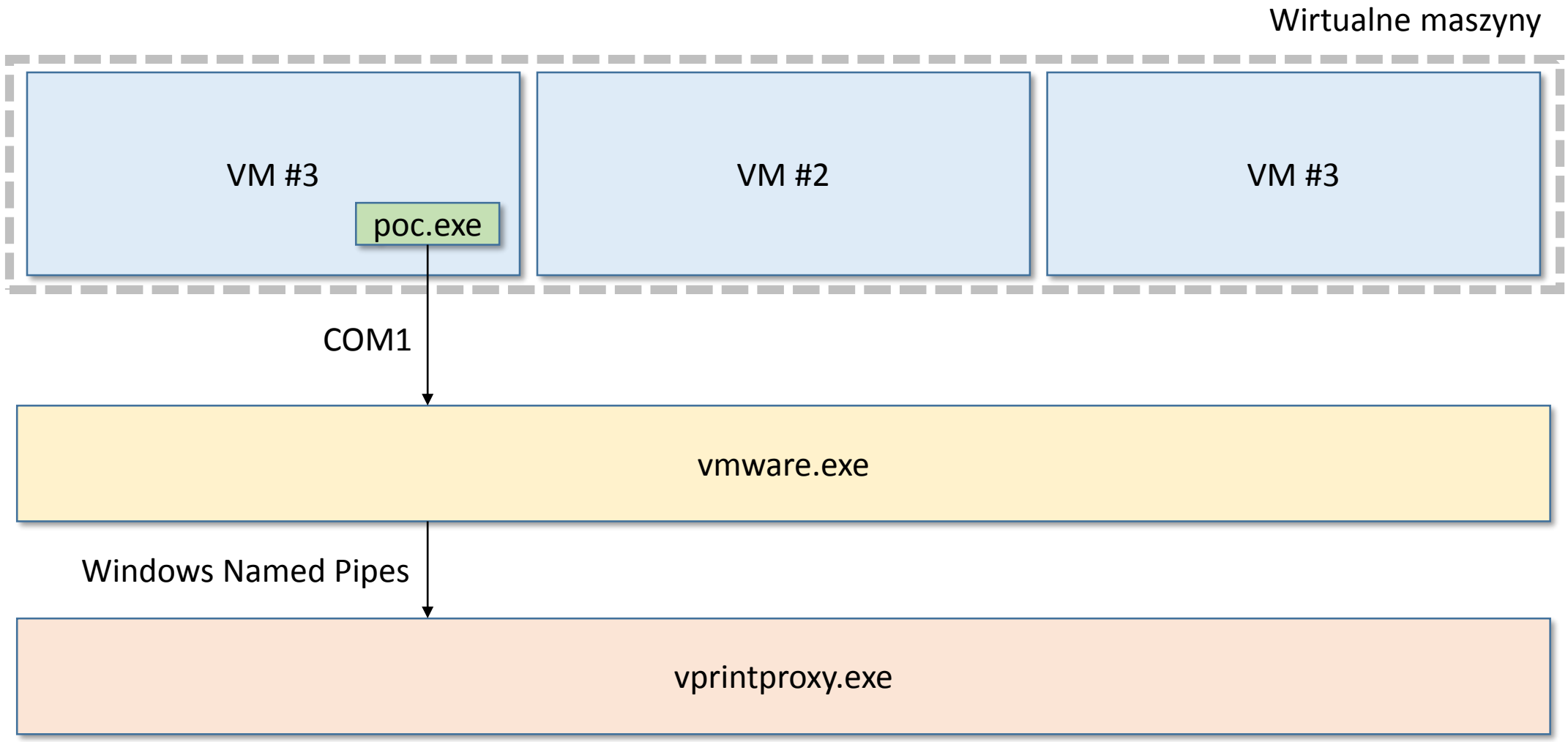
- Pliki EMF są również wykorzystywane w procesie buforowania wydruków (*print spooling*).
- Otwiera nam to drogę do nowych wektorów ataku związanych z tym formatem, jak np. sterowniki drukarek (i powiązane oprogramowanie).
- Jednym z ciekawych celów może być VMware Workstation.

Virtual printers

- Opcja pozwalająca wirtualnym maszynom na drukowanie dokumentów przy użyciu drukarek zainstalowanych na systemie-hoście.
- Potencjalna droga ucieczki z systemu-gościa do systemu nadzorującego.
- Zgodnie z moją wiedzą opcja ta była włączona domyślnie jeszcze w 2015 r., ale niedawno została wyłączona (prawdopodobnie w związku z błędami zgłoszonymi przez Kostyę Kortchinsky).



Architektura



Architektura

- Atakowanym procesem jest vprintproxy.exe działający na hoście.
  - Odbiera dane wysłane przez nieuprzywilejowany proces w wirtualizowanym systemie w praktycznie niezmienionej formie.
  - Obiecujący kanał komunikacji.
- Dane powinny być przesyłane w formacie EMFSPOOL.
  - Podobny do EMF, z dodatkową możliwością zdefiniowania własnych czcionek w wielu różnych formatach.

TPView

- Uściślając, sama obsługa EMF znajduje się w pliku TPview.dll.
  - Ta oraz kilka innych dołączonych bibliotek związanych z drukowaniem są rozwijane przez zewnętrzną firmę, ThinPrint.
- W większości odwołuje się bezpośrednio do GDI, ale implementuje również obsługę kilku własnych, nieudokumentowanych rekordów.
  - W przeszłości było tam wiele błędów, ale Kostya znalazł i zgłosił (prawie) wszystkie z nich.
  - Przejrzałem kod samemu po raz drugi, znalazłem dwa raczej niegroźne błędy (#848 i #849), i to wszystko.

Dekodowanie JPEG2000

- Odkryłem za to jeden ciekawy rekord EMF, który wydawał się zupełnie niezbadany.
  - ID = 0x8000.
  - Na podstawie pomocniczych łańcuchów tekstowych można się domyśleć, że jest związany z dekodowaniem formatu JPEG2000.
- Osobiście nie jestem ekspertem od JPEG2K, a kod jest bardzo skomplikowany i trudny do ręcznej analizy.
- Czas na fuzzing?

Podejście do fuzzingu

- Najlepszy fuzzing: na Linuksie, w dużej skali, z AddressSanitizerem i informacją o pokryciu kodu.
- Po dalszej analizie udało się odkryć, że sam dekoder JPEG2000 jest autorstwa jeszcze innego producenta, LuraTech.
  - Komercyjna licencja, brak publicznie dostępnego kodu źródłowego.
- Czy w takim razie utknęliśmy z TPview.dll używanym przez VMware Workstation?
  - Fuzzing takiej konfiguracji też jest możliwy, ale trudniejszy, wolniejszy, i gorszy jakościowo.

Dalsze badania

- Kolejne godziny badań pozwoliły ustalić, że ten sam producent wypuścił również darmowy plugin do obsługi formatu JPEG2000 dla popularnego programu IrfanView.
  - JPEG2000.DLL.
  - Po pobieżnej analizie widać, że kod obu bibliotek jest faktycznie bardzo podobny (o ile nie identyczny).
- Interfejs wtyczki jest niezwykle prosty, i sprowadza się do wywołania jednej eksportowanej funkcji o następującej deklaracji.

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

Zbliżamy się do celu…

- Dzięki temu, jesteśmy w stanie uruchomić szybki fuzzing w obrębie jednego procesu na Windowsie, bez konieczności uruchamiania VMware.
  - Prosty program ładujący bibliotekę DLL i wywołujący odpowiednią funkcję zajmuje <50 linii kodu.
- Mimo wszystko, wolałbym jednak uruchomić fuzzing na Linuksie...

Fuzzing biblioteki DLL na Linuksie

- W zasadzie czemu nie?
- Preferowany adres bazowy to 0x10000000, który jest z reguły dostępny w przestrzeni adresowej.
  - Obsługa relokacji nie jest konieczna; wystarczy podmapować sekcje pod odpowiednimi adresami, z poprawnymi prawami dostępu.
- Inne konieczne czynności:
  - Rozwiązywanie niezbędnych importów.
  - Odnalezienie adresu interesującej nas eksportowanej funkcji.
  - Wywołanie owej funkcji w celu uruchomienia dekodowania.
- Powinno zadziałać!

Rozwiązywanie importów

- Tabela importów może tu być jedyną problematyczną kwestią.
  - W szczególności funkcje WinAPI niedostępne na Linuksie.
- Nasza konkretna biblioteka importuje z ADVAPI32, KERNEL32, MSVCRT, SHELL32 i USER32.
  - Importy z MSVCRT mogą zostać przekierowane bezpośrednio do libc.
  - Wszystkie pozostaje musiałyby być albo wyemulowane, albo chociaż wypełnione pustymi funkcjami.

Importy KERNEL32

- Tylko trzy używane funkcje WinAPI: `GlobalAlloc`, `GlobalLock` i `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;
}
```

Brakujące importy libc

- Znaleźliśmy również dwa importy specyficzne dla MSVCRT, które musiały zostać przepisane:

```
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);
}
```

To działa!

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

Uruchamianie fuzzingu

- Jako korpusu wejściowego użyłem zbioru plików wygenerowanego wcześniej na podstawie otwartej implementacji dekodera tego formatu.
- Strategia mutacji została dobrana tak, by odpowiadać zasadzie 50/50 współczynnika pomyślnego/błędnego przetwarzania danych wejściowych.
- Pozostawiłem działający fuzzer na kilka dni, by wrócić do...
  - ... 186 *crashy* z unikalnymi stosami wywołań funkcji.

Reprodukcja crashy

- Pamiętajmy, że wszelkie znalezione crashe znajdują się w bibliotece wtyczki do IrfanView, a nie w VMware Workstation.
- Proces vprintproxy.exe jest bardzo łatwy w interakcji: tworzy nazwany obiekt *pipe* i odczytuje z niego dokładnie te same dane, które są przesyłane w systemie-gościu na port COM1.
  - Ponownie możemy przetestować pliki wejściowe bez uruchamiania maszyny wirtualnej.
- Warto włączyć mechanizm PageHeap dla testowanego procesu, w celu umożliwienia lepszego wykrywania i deduplikacji błędów.

Ostateczne wyniki

Instrukcja	Powód
`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

Instrukcja	Powód
`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

Ostateczne wyniki

- Crashe w **39** unikalnych instrukcjach.
  - Wiele z nich wystąpiło wewnątrz uniwersalnych funkcji takich jak `memcpy()`, więc nie jest to w pełni dokładna metryka.
  - Szybka klasyfikacja: **18** niskie zagrożenie, **15** średnie zagrożenie, **6** wysokie zagrożenie.
- Wszystkie zgłoszone do VMware 15 czerwca.
- Naprawione jako część biuletynu VMSA-2016-0014 13 sierpnia (w ciągu 90 dni).

Podsumowanie

Podsumowanie

- Pliki metafile są na tyle skomplikowane i interesujące, że ich badaniom warto poświęcić więcej czasu.
  - Wspierane przez wiele istotnych wektorów ataku.
- Można dowiedzieć się z nich ciekawych rzeczy na temat pewnych nieudokumentowanych interfejsów w systemie (np. NamedEscape).
- Jak zwykle okazuje się, że im starszy i słabiej zbadany format/implementacja, tym lepiej dla osoby poszukującej błędów.
- Inspiracja pracą innych badaczy popłaca. ☺
- Odpowiednie narzędzie do odpowiedniego zadania – ręczna analiza kodu vs fuzzing.

Dziękuję!

@j00ru

http://j00ru.vexillium.org/

j00ru.vx@gmail.com