The Life And Death of Kernel Object Abuse

Saif ElSherei (0x5A1F) & Ian Kronquist

Microsoft

Who?

@Saif_Sherei Senior Security Software Engineer @ MSRC

@IanKronquist Software Engineer on the Windows Device Group Security Team

There’s Definitely a Method to Madness (Why?)

Microsoft

Attack Chain

Trigger some exploitable bug, UAF, OOB write, Integer overflow

→

Gain higher exploit primitive by abusing certain objects, to gain arbitrary kernel memory read/write

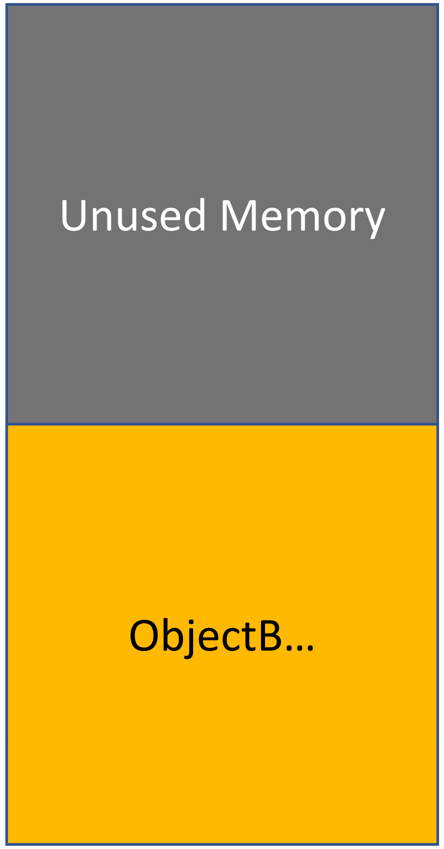
→

Use arbitrary kernel memory read/write to steal a SYSTEM process token, and assign it to the current process

What ??

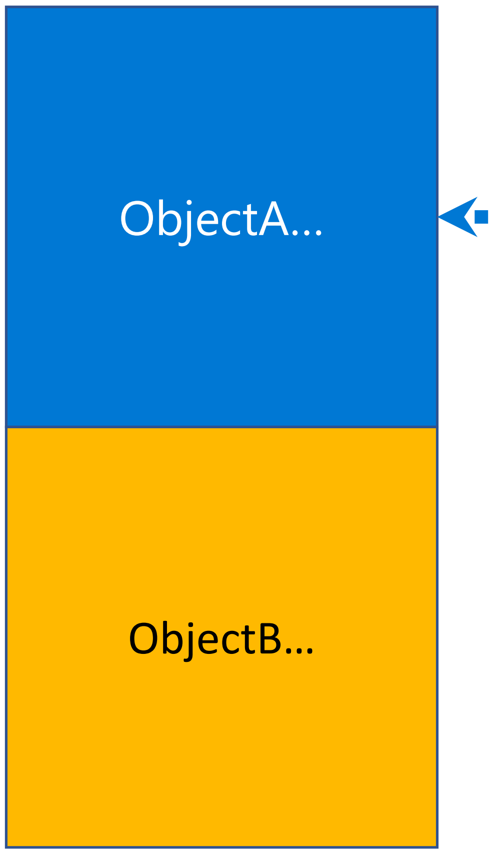
Microsoft

Memory Corruption - UAF



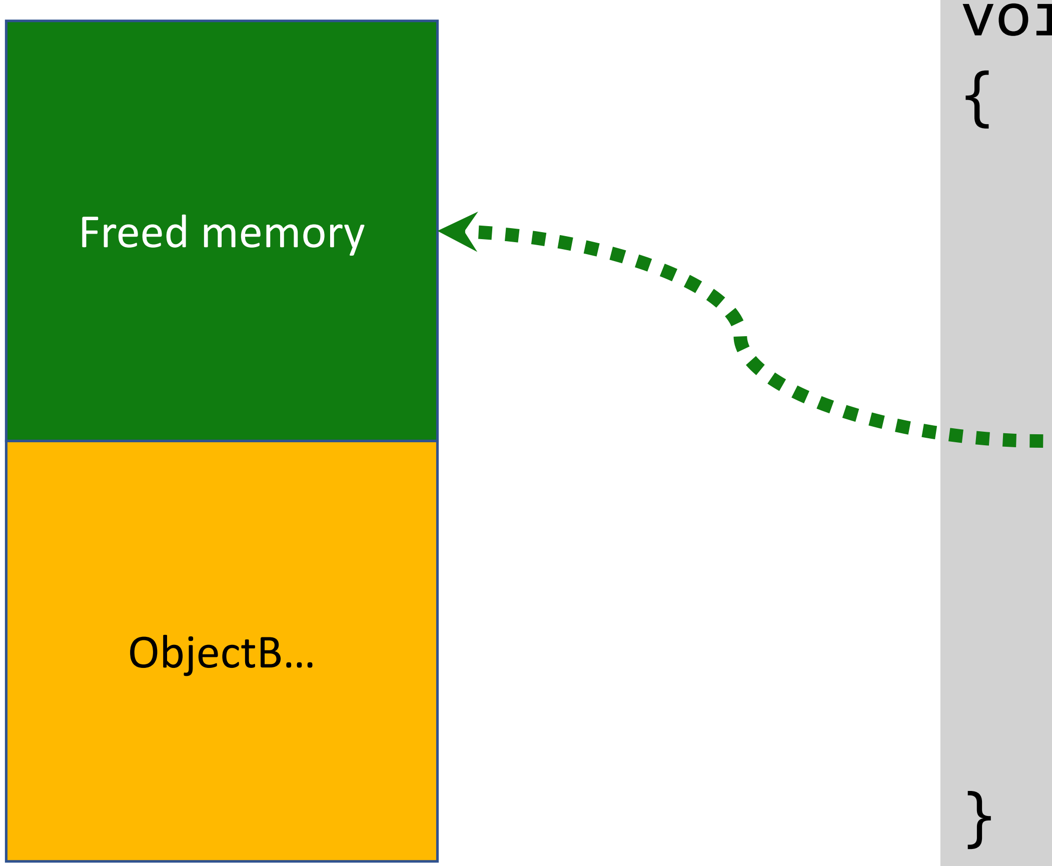

```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    …
    ObjectA.B = 0x41414141;
    …
    return;
}
```

Memory Corruption – UAF - Allocate



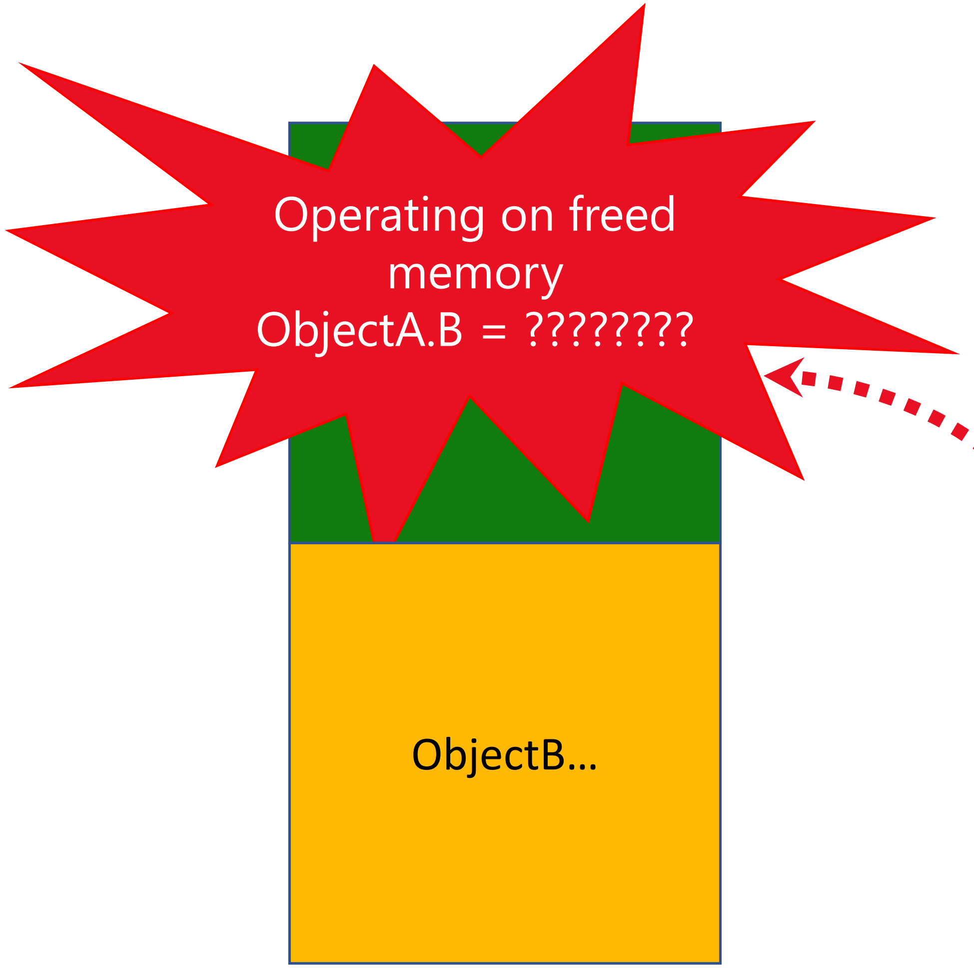

```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    …
    ObjectA.B = 0x41414141;
    …
    return;
}
```

Memory Corruption – UAF - Free



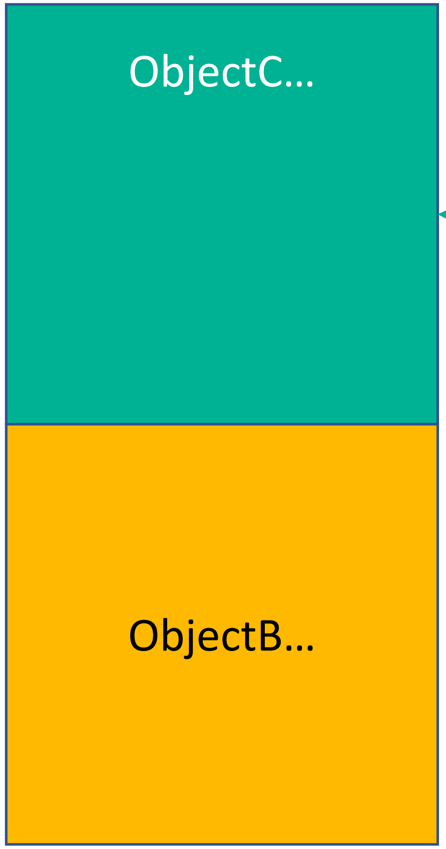

```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    …
    ObjectA.B = 0x41414141;
    …
    return;
}
```

Memory Corruption – UAF - Use




```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    …
    ObjectA.B = 0x41414141;
    …
    return;
}
```

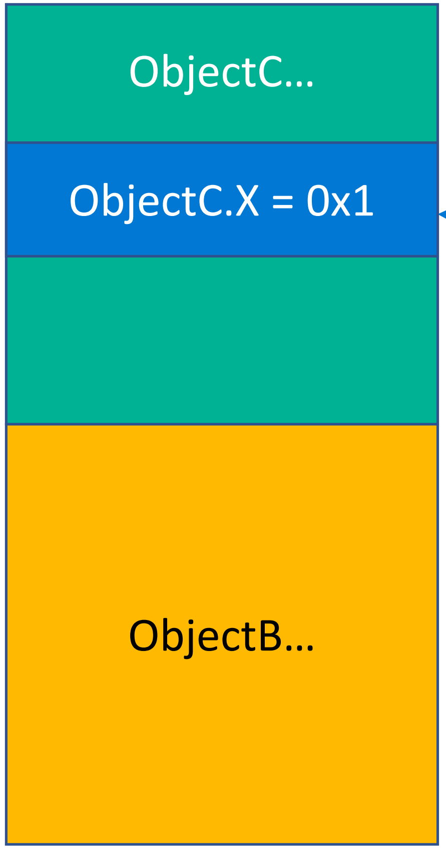
Memory Corruption – UAF - Exploitation



Replace freed ObjectA with a new ObjectC of the same size and allocated to the same heap.

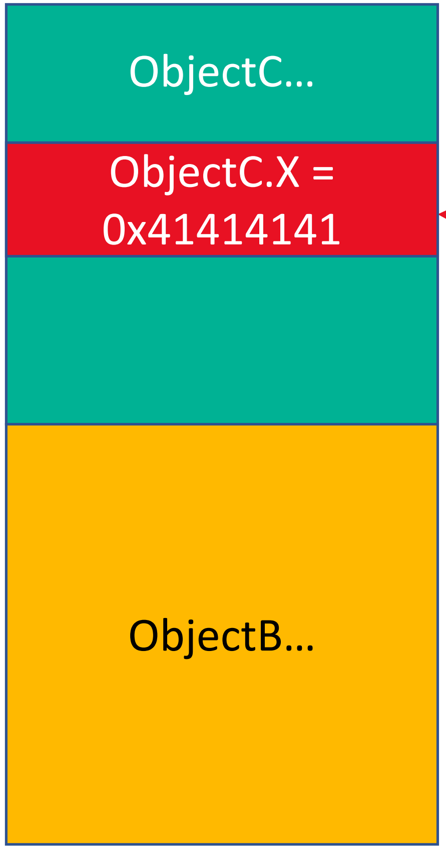
```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    NewObj ObjectC = new NewObj();
    ObjectC.X = 0x1;
    ObjectA.B = 0x41414141;
    printf(“%x”,ObjectC.X);
    …
    return;
}
```

Memory Corruption – UAF - Exploitation




```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    NewObj ObjectC = new NewObj();
    ObjectC.X = 0x1;
    ObjectA.B = 0x41414141;
    printf("%x",ObjectC.X);
    …
    return;
}
```

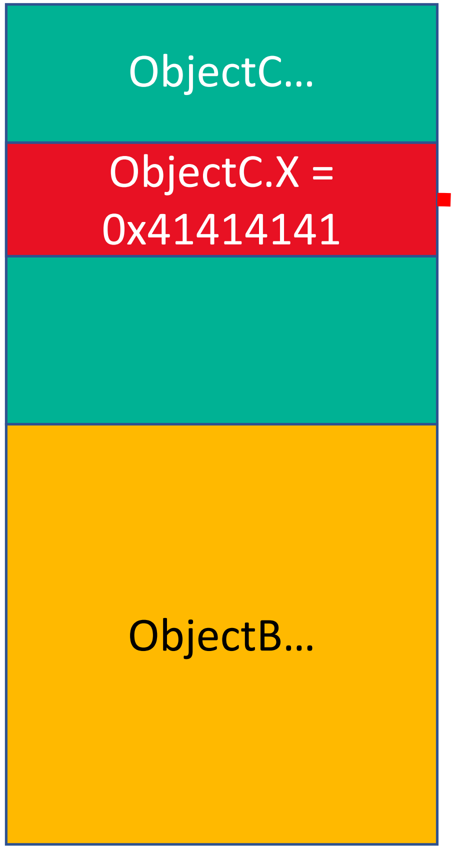
Memory Corruption – UAF - Exploitation



When ObjectA gets used after its freed it will corrupt ObjectC members.

```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)
    {
        Free(ObjectA);
    }
    NewObj ObjectC = new NewObj();
    ObjectC.X = 0x1;
    ObjectA.B = 0x41414141;
    printf("%x",ObjectC.X);
    …
    return;
}
```

Memory Corruption – UAF - Exploitation



Corrupted ObjectC.X =
ObjectA.B = 0x41414141

```
void SimpleUafFunction()
{
    …
    Object ObjectA =  new Object();
    ….
    If (condition == 0)

        Free(ObjectA);

    NewObj ObjectC = new NewObj();
    ObjectC.X = 0x1;
    ObjectA.B = 0x41414141;
    printf("%x",ObjectC.X);
    …
    return;
}
```

Memory Corruption – x86 Integer Overflow

$\cancel{0xF \ldots x81}$

Actually

$$0xFFFFFF80 + 0x81 = 0x0100000001$$

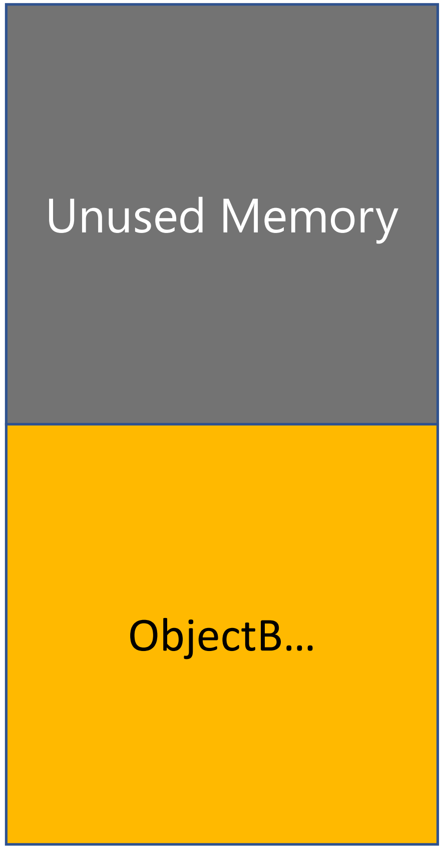
$0x0100000001$ > 32-bit (4 bytes) wide registers

Integer will be truncated to fit register size

Most significant byte ignored 0x01

$$0xFFFFFF80 + 0x81 = 0x00000001$$

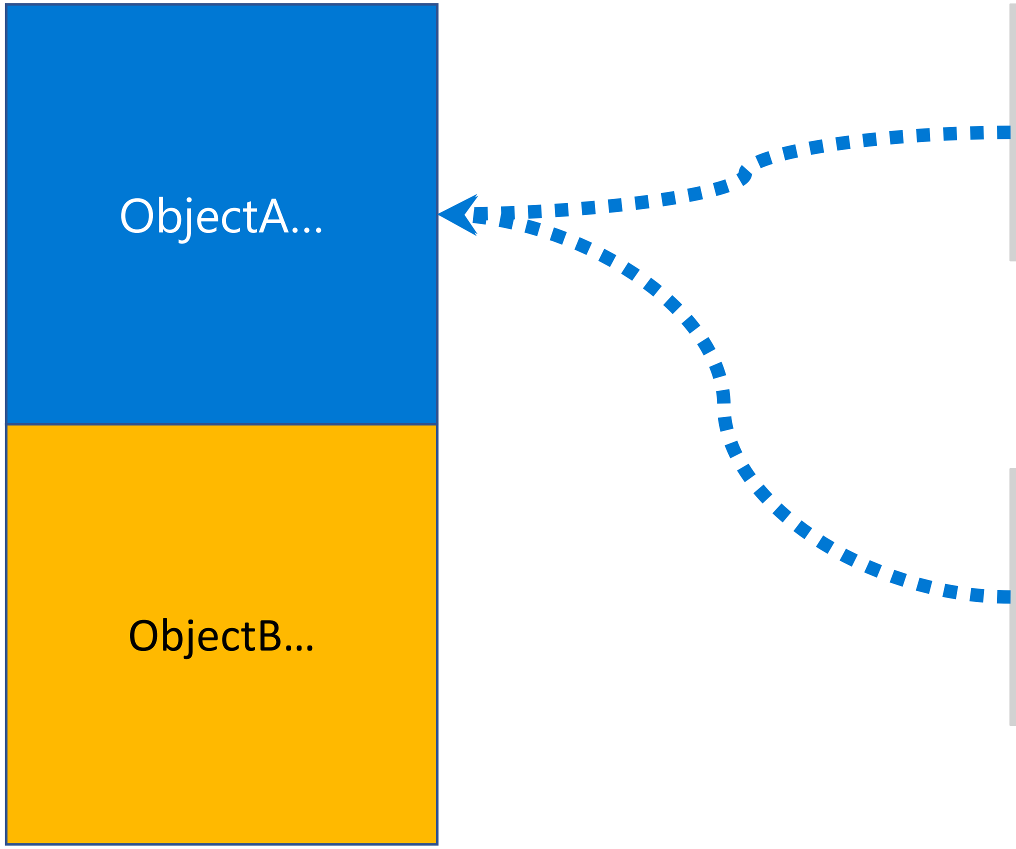
Memory Corruption – Linear Overflow




```
Scenario A:
objPtr = AllocateObject(sz_overflow);
memcpy(objPtr,src,sz_original);
```

```
Scenario B:
objPtr = AllocateObject(sz_fixed);
memcpy(objPtr,src,sz_usersupplied);
```

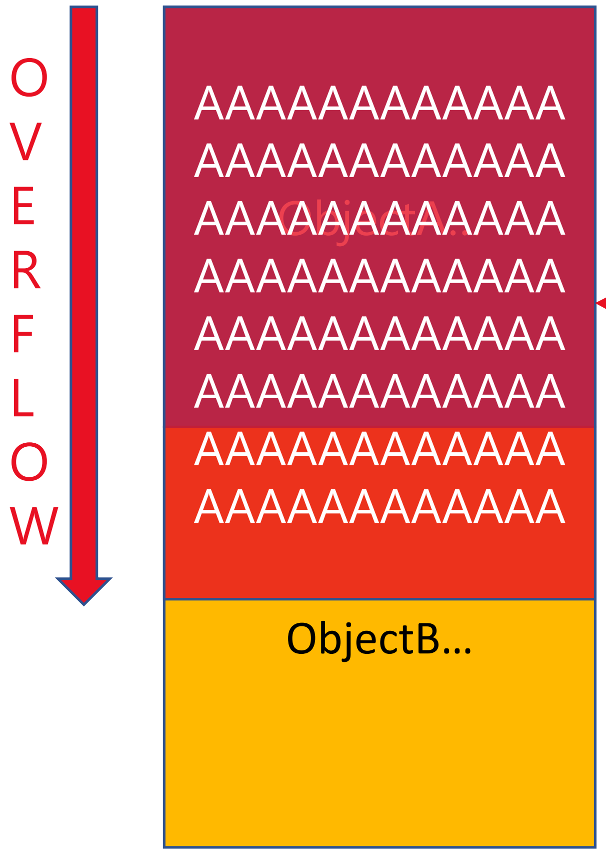
Memory Corruption – Linear Overflow




```
Scenario A:
objPtr = AllocateObject(sz_overflow);
memcpy(objPtr,src,sz_original);
```

```
Scenario B:
objPtr = AllocateObject(sz_fixed);
memcpy(objPtr,src,sz_usersupplied);
```

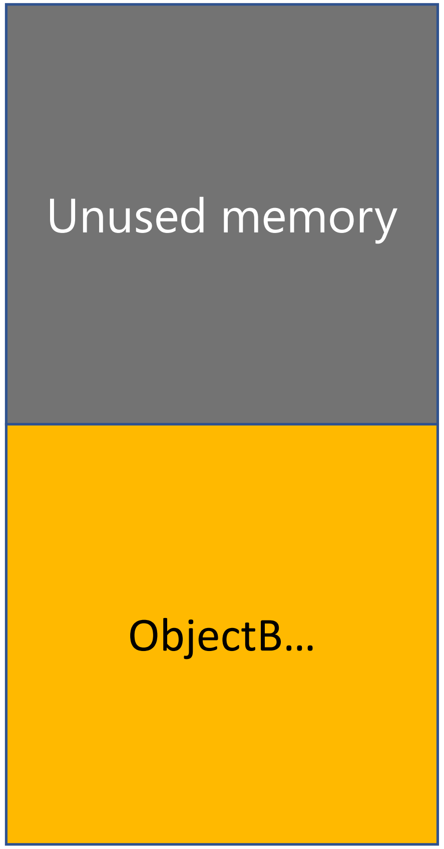
Memory Corruption – Linear Overflow




```
Scenario A:
objPtr = AllocateObject(sz_overflow);
memcpy(objPtr,src,sz_original);
```

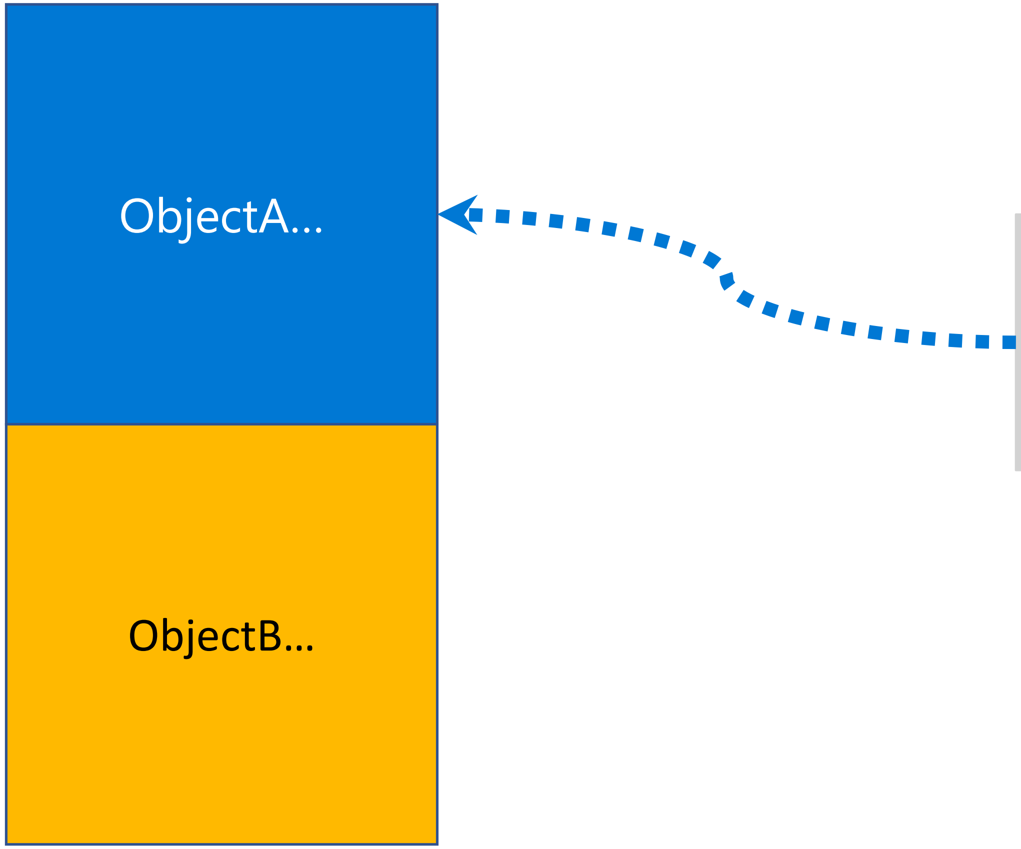
```
Scenario B:
objPtr = AllocateObject(sz_fixed);
memcpy(objPtr,src,sz_usersupplied);
```

Memory Corruption – OOB Write



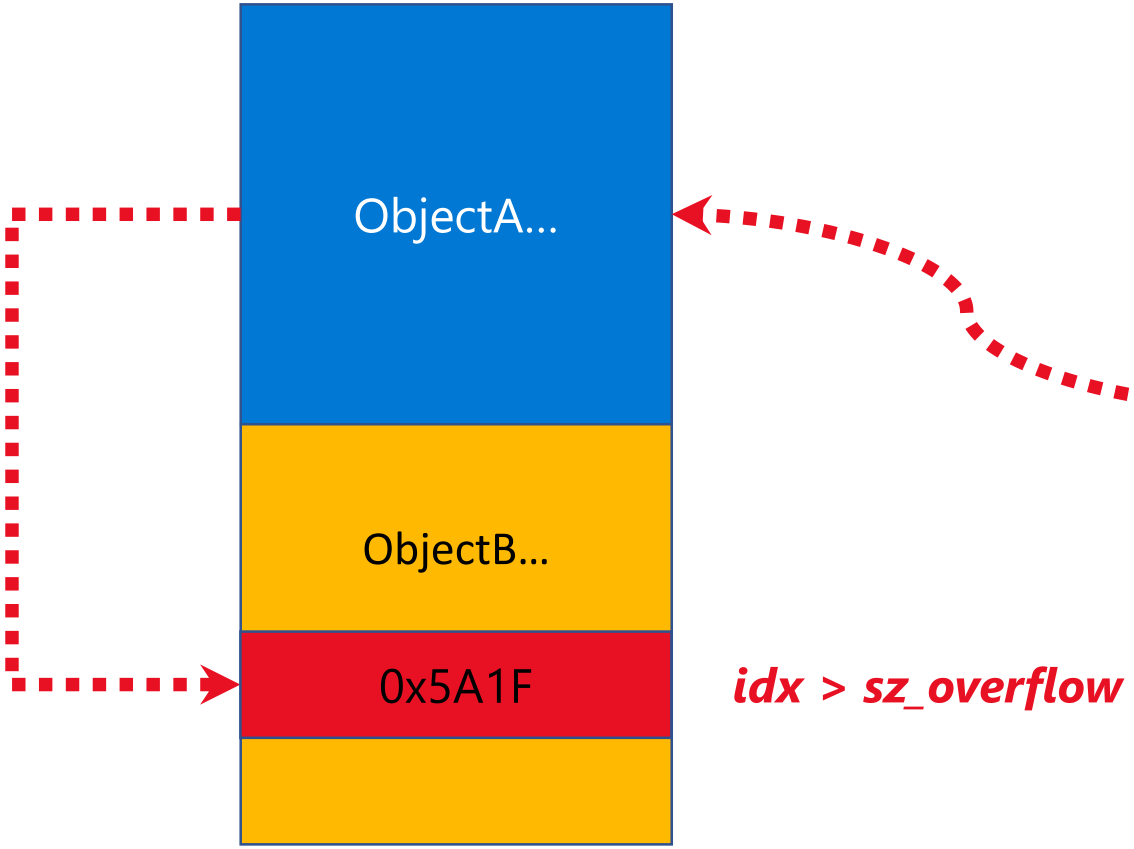

```
Scenario:
objPtr = AllocateObject(sz_overflow);
objPtr[idx > sz_overflow] = 0x5A1F;
```

Memory Corruption – OOB Write




```
Scenario:
objPtr = AllocateObject(sz_overflow);
objPtr[idx > sz_overflow] = 0x5A1F;
```

Memory Corruption – OOB Write




```
Scenario:
objPtr = AllocateObject(sz_overflow);
objPtr[idx > sz_overflow] = 0x5A1F;
```

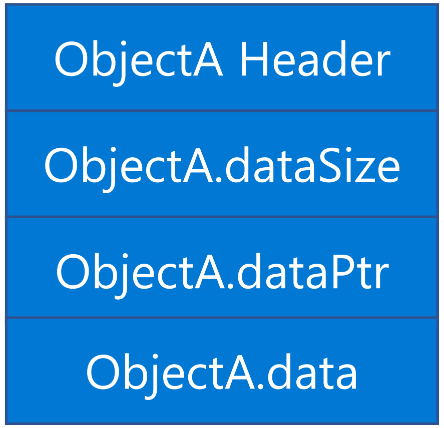
Memory Corruption – OOB OF Exploitation

- Get Kernel 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 before one of the user controlled objects.
- Use overflow or OOB write to corrupt interesting members of the user controlled object.

The Life of Kernel Object Abuse (How ??)

Microsoft

Abusing Objects For Fun & Profit



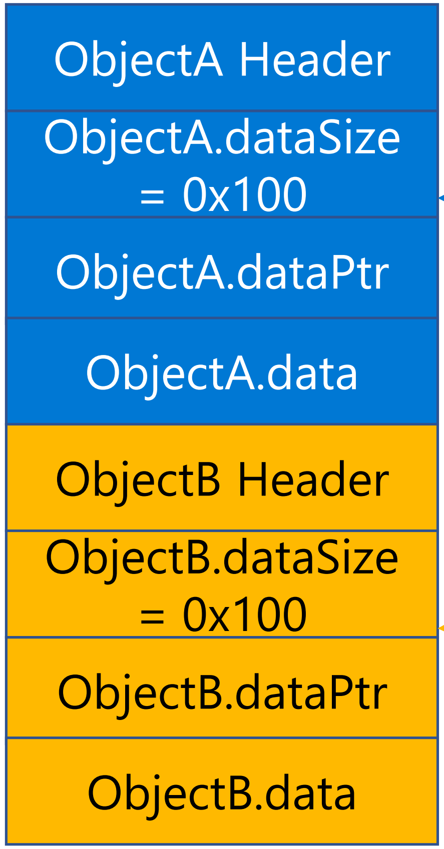
Interesting Objects members:

- Size member (allows relative memory r/w)
- Pointer to data (allows arbitrary memory r/w)

Interesting Functions:

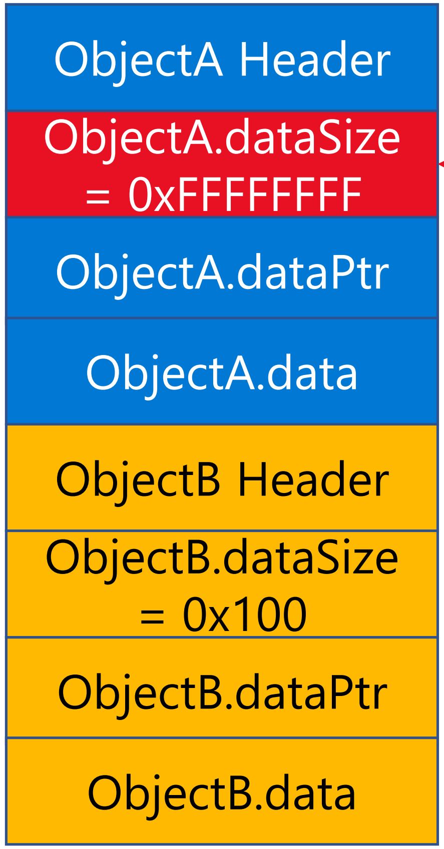
- GetData(…)
- SetData(…)

Abusing Objects For Fun & Profit




```
void Exploit()
{
    …
    Object ObjectA =  new Object();
    Object ObjectB =  new Object();
    …
    ExploitChangeSize(ObjectA,
                               0xFFFFFFFF);
    …
    BYTE * buff = GetData(ObjectA);
    …
    SetData(ObjectA, 0x41414141, idx, sz);
    BYTE * out = GetData(ObjectB);
    return;
}
```

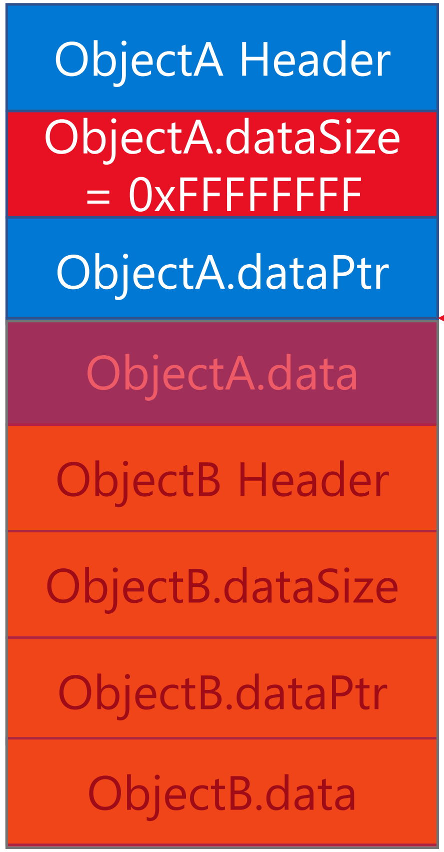
Abusing Objects For Fun & Profit



Exploit UAF or Integer issue, to corrupt the ObjectA.dataSize member

```
void Exploit()
{
    …
    Object ObjectA =  new Object();
    Object ObjectB =  new Object();
    …
    ExploitChangeSize(ObjectA,
                          0xFFFFFFFF);
    …
    BYTE * buff = GetData(ObjectA);
    …
    SetData(ObjectA, 0x41414141, idx, sz);
    BYTE * out = GetData(ObjectB);
    return;
}
```

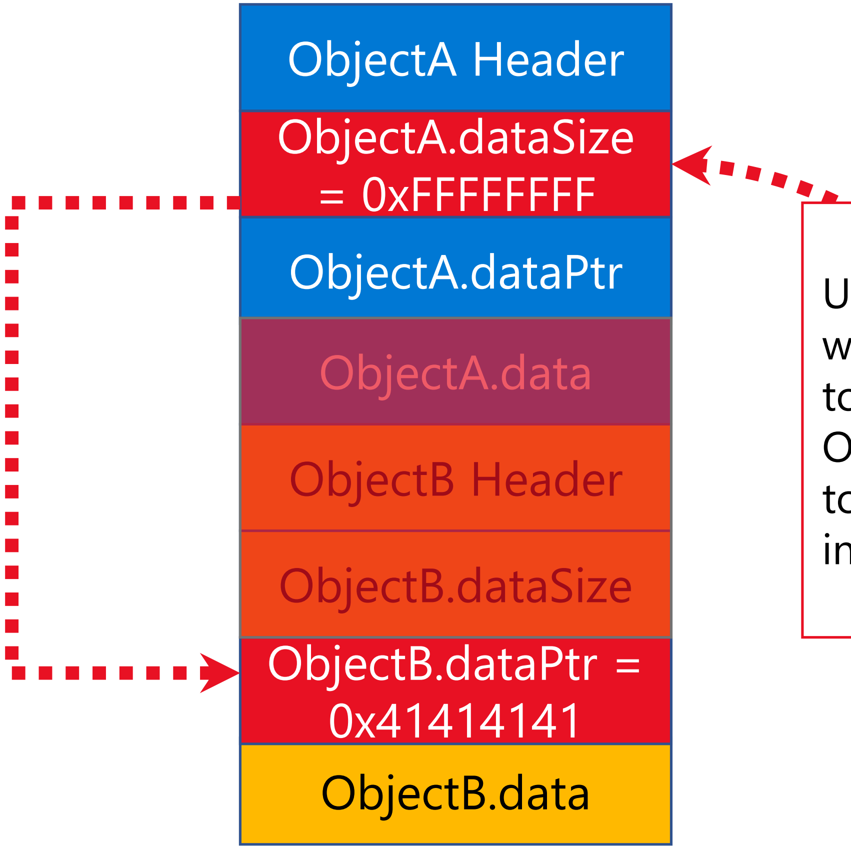
Abusing Objects For Fun & Profit



Read data up to corrupted size of 0xFFFFFFFF (4GB) gaining memory read/write **relative** to ObjectA.dataPtr

```
void Exploit()
{
    …
    Object ObjectA =  new Object();
    Object ObjectB =  new Object();
    …
    ExploitChangeSize(ObjectA,
                               0xFFFFFFFF);
    …
    BYTE * buff = GetData(ObjectA);
    …
    SetData(ObjectA, 0x41414141, idx, sz);
    BYTE * out = GetData(ObjectB);
    return;
}
```

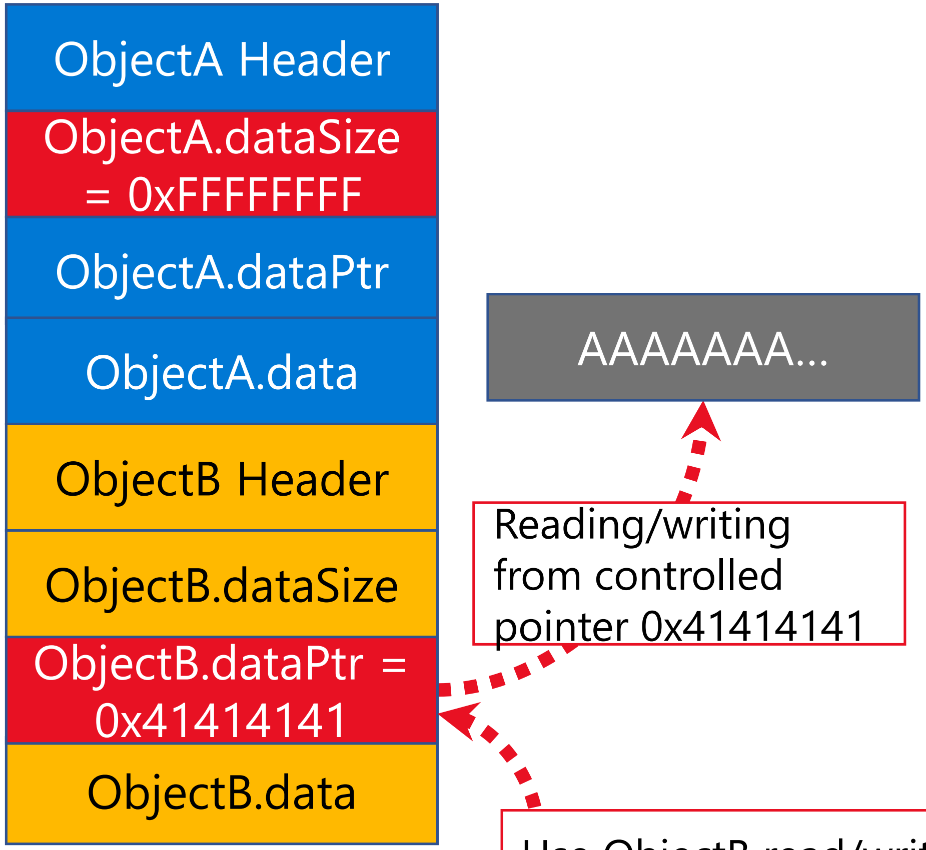
Abusing Objects For Fun & Profit



Use ObjectA write functions to overwrite ObjectB.dataPtr to any address in memory.

```
void Exploit()
{
    …
    Object ObjectA =  new Object();
    Object ObjectB =  new Object();
    …
    ExploitChangeSize(ObjectA,
                             0xFFFFFFFF);
    …
    BYTE * buff = GetData(ObjectA);
    …
    SetData(ObjectA, 0x41414141, idx, sz);
    BYTE * out = GetData(ObjectB);
    return;
}
```

Abusing Objects For Fun & Profit




```
void Exploit()
{
    …
    Object ObjectA =  new Object();
    Object ObjectB =  new Object();
    …
    ExploitChangeSize(ObjectA,
                          0xFFFFFFFF);
    …
    BYTE * buff = GetData(ObjectA);
    …
    SetData(ObjectA, 0x41414141, idx, sz);
    BYTE * out = GetData(ObjectB);
    return;
}
```

Use ObjectB read/write functions to read/write from controlled memory pointer gaining **arbitrary** memory read/write

Win32k Memory

- Desktop Heap (NTUSER)
  - Window management related objects.
  - Window(s) objects, Menus, Classes, etc …
  - Objects allocated/free-ed using RtlAllocateHeap/RtlFreeHeap.
- Paged Session Pool (NTGDI)
  - GDI related objects.
  - GDI bitmaps, palettes, brushes, DCs, lines, regions, etc …
  - Objects usually allocated/free-ed using ExAllocatePoolWithTag/ExFreePoolWithTag.
- Non-Paged Session Pool (not in scope for this presentation)

Statistics

Object Type	MSRC Count	% MSRC Win32k UAF surface	Type location	Release
Surface	11	12.22	GDI	RS3
tagWND	9	10	USER	RS4
tagCURSOR	8	8.89	USER	RS4
tagMENU	7	7.78	USER	RS4
tagCLS	4	4.44	USER	RS4
tagpopupmenu	4	4.44	USER	RS4
Palette	2	2.22	GDI	RS4
Pen + Brush	2	2.22	GDI	RS4
RFFont	1	1.11	GDI	RS4
Path	0	N/A	GDI	RS4

Abusing Window Objects tagWnd

Abusing Window Objects tagWnd

```
1: kd> dt win32kbase!tagwnd -b
   +0x000 head                 : _THRDESKHEADSHARED
      +0x000 h                    : Ptr64
      +0x008 cLockObj             : Uint4B
      +0x010 pti                  : Ptr64
      +0x018 rpdesk               : Ptr64
      +0x020 pSelf                : Ptr64
      +0x028 pSharedPtr           : Ptr64
      +0x030 pOffset              : Uint8B
------SNIPPED---------------------
   +0x0d8 hrgnClip             : Ptr64
   +0x0e0 hrgnNewFrame         : Ptr64
   +0x0e8 strName              : _LARGE_UNICODE_STRING
      +0x000 Length               : Uint4B
      +0x004 MaximumLength        : Pos 0, 31 Bits
      +0x004 bAnsi                : Pos 31, 1 Bit
      +0x008 Buffer               : Ptr64
   +0x0f8 cbwndExtra           : Int4B
   +0x0fc cbWndServerExtra     : Uint4B
   +0x100 spwndLastActive      : Ptr64
   +0x108 hImc                 : Ptr64
   +0x110 dwUserData           : Uint8B
-------SNIPPED---------------------
   +0x180 pExtraBytes          : Uint8B
   +0x188 pServerExtraBytes : Ptr64
```

Abusing Window Objects tagWnd- Allocation

Syntax

C++

```
HWND WINAPI CreateWindow(
  _In_opt_ LPCTSTR   lpClassName,
  _In_opt_ LPCTSTR   lpWindowName,
  _In_     DWORD     dwStyle,
  _In_     int       x,
  _In_     int       y,
  _In_     int       nWidth,
  _In_     int       nHeight,
  _In_opt_ HWND      hWndParent,
  _In_opt_ HMENU     hMenu,
  _In_opt_ HINSTANCE hInstance,
  _In_opt_ LPVOID    lpParam
);
```

Syntax

C++

```
HWND WINAPI CreateWindowEx(
  _In_     DWORD     dwExStyle,
  _In_opt_ LPCTSTR   lpClassName,
  _In_opt_ LPCTSTR   lpWindowName,
  _In_     DWORD     dwStyle,
  _In_     int       x,
  _In_     int       y,
  _In_     int       nWidth,
  _In_     int       nHeight,
  _In_opt_ HWND      hWndParent,
  _In_opt_ HMENU     hMenu,
  _In_opt_ HINSTANCE hInstance,
  _In_opt_ LPVOID    lpParam
);
```

https://msdn.microsoft.com/en-us/library/windows/desktop/ms632679(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms632680(v=vs.85).aspx

Abusing Window Objects tagWnd - Free

Syntax

C++

```
BOOL WINAPI DestroyWindow(
  _In_ HWND hWnd
);
```

Parameters

hWnd [in]

Type: **HWND**

A handle to the window to be destroyed.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms632682(v=vs.85).aspx

Abusing Window Objects tagWnd– Read Data

C++

```
LONG WINAPI GetWindowLong(
  _In_ HWND hWnd,
  _In_ int  nIndex
);
```

C++

```
LONG_PTR WINAPI GetWindowLongPtr(
  _In_ HWND hWnd,
  _In_ int  nIndex
);
```

C++

```
int WINAPI InternalGetWindowText(
  _In_  HWND   hWnd,
  _Out_ LPWSTR lpString,
  _In_  int    nMaxCount
);
```

GetWindowLongPtr:
- Reads Long at index < cbwndExtra from ExtraBytes.

InternalGetWIndowText:
- Reads Length <= MaximumLength string from strName buffer.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms633584(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms633523(v=vs.85).aspx

Abusing Window Objects tagWnd – Write Data

Syntax

C++

```cpp
LONG WINAPI SetWindowLong(
  _In_ HWND hWnd,
  _In_ int  nIndex,
  _In_ LONG dwNewLong
);
```

C++

```cpp
LONG_PTR WINAPI SetWindowLongPtr(
  _In_ HWND     hWnd,
  _In_ int      nIndex,
  _In_ LONG_PTR dwNewLong
);
```

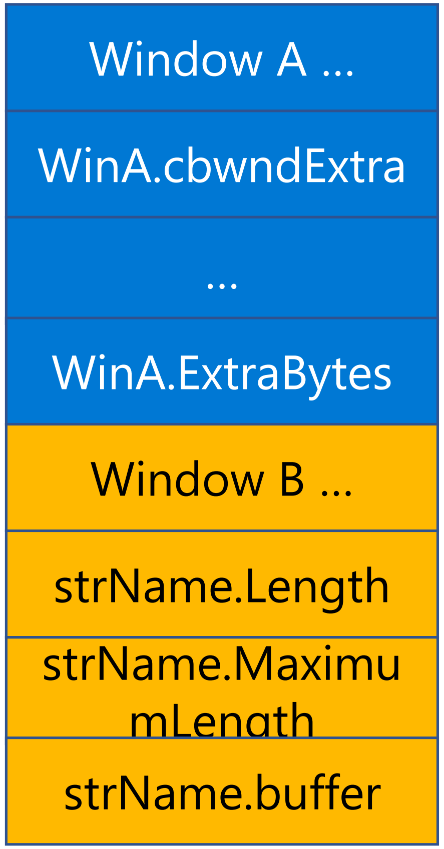
SetWindowLongPtr:
- Write Long at index < cbwndExtra into ExtraBytes.

NtUserDefSetText:
- Writes up to Length <= MaximumLength string from strName buffer.

```cpp
BOOL NtUserDefSetText( HWND hWnd, PLARGE_STRING pstrText );
```

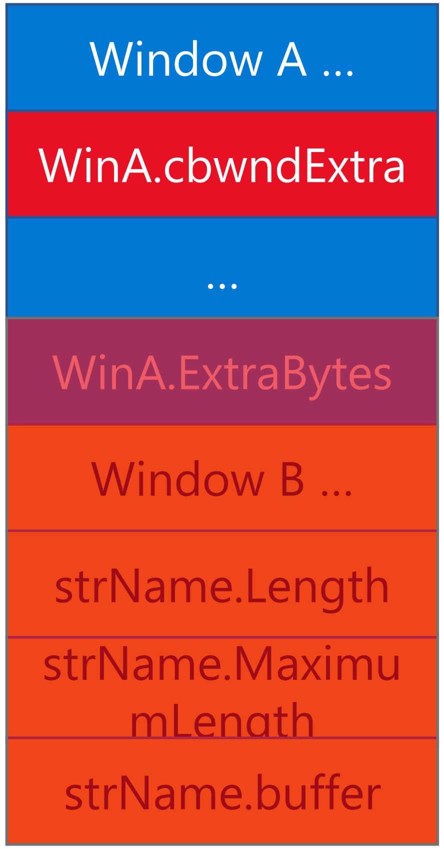
https://msdn.microsoft.com/en-us/library/windows/desktop/ms633591(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms644898(v=vs.85).aspx

Abusing Window Objects tagWnd – Exploitation



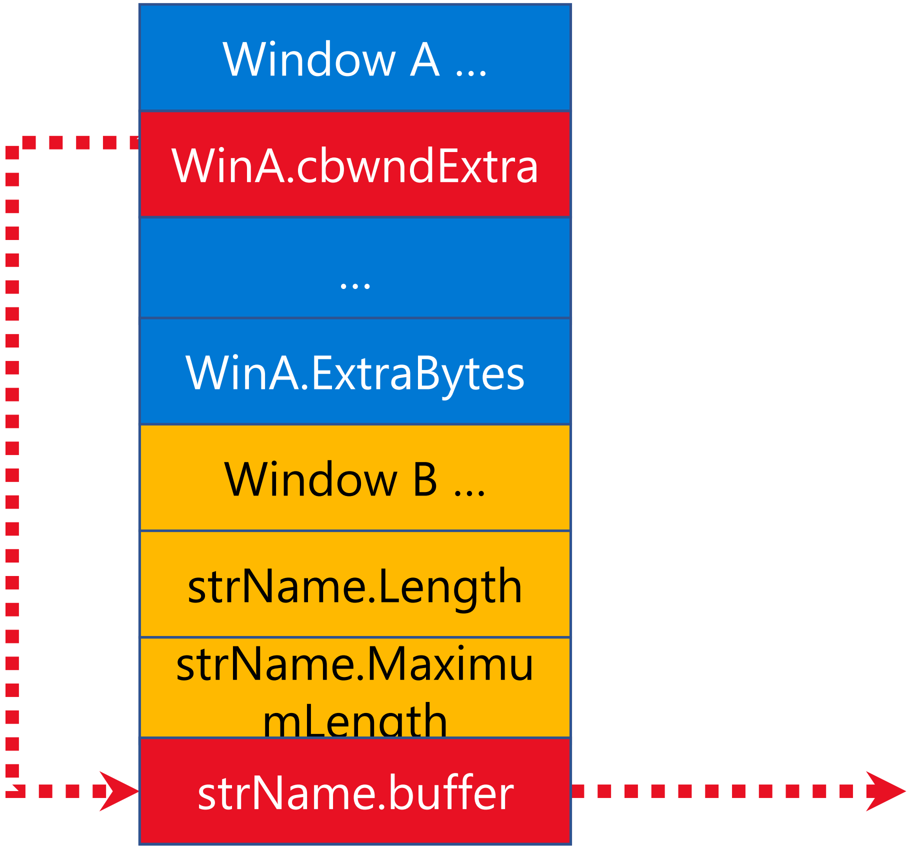
- Window A & Window B are two adjacent Window objects.

Abusing Window Objects tagWnd – Exploitation



- Use a kernel bug to corrupt Window A cbwndExtra member.
- This will extend the Window A extra data, gaining memory read/write relative to WindowA.ExtraBytes into the adjacent Window B.
- Window A will be the manager object that will be used to set the pointer on Window B to be read/write from.

Abusing Window Objects tagWnd – Exploitation



- Window B will be the worker object.
- Use Window A relative r/w to overwrite (set) Window B strName.buffer to any location in kernel memory.
- Using Window B read/write functions, allows arbitrary kernel memory read/write.

Read/write to/from any arbitrary kernel memory location pointed to by Window B strName.buffer

Abusing Bitmaps _SURFOBJ

First disclosed by KeenTeam @k33nTeam (2015)

Heavily detailed & analysed by Nico Economou @NicoEconomou and Diego Juarez (2015/2016)

Microsoft

Abusing Bitmaps _SURFOBJ

Object type _SURFOBJ

PoolTag Gh?5, Gla5

x86

SURFOBJ

```
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 Bitmaps _SURFOBJ - Allocation

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

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.

```
HBITMAP bmp =   CreateBitmap(
                0x3A3,   //nWidth
                1,       //nHeight
                1,       //cPlanes
                32,      //cBitsPerPel
                NULL);   // lpvBits
```

https://msdn.microsoft.com/en-us/library/windows/desktop/dd183485(v=vs.85).aspx

Abusing Bitmaps _SURFOBJ - Free

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

Parameters

hObject [in]
    A handle to a logical pen, brush, font, bitmap, region, or palette.

https://msdn.microsoft.com/en-us/library/windows/desktop/dd183539(v=vs.85).aspx

Abusing Bitmaps _SURFOBJ – Read Data

```
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.

Reads up to sizlBitmap data, from address pointed to by pvScan0.

https://msdn.microsoft.com/en-us/library/windows/desktop/dd144850(v=vs.85).aspx

Abusing Bitmaps _SURFOBJ – Write Data

```
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.

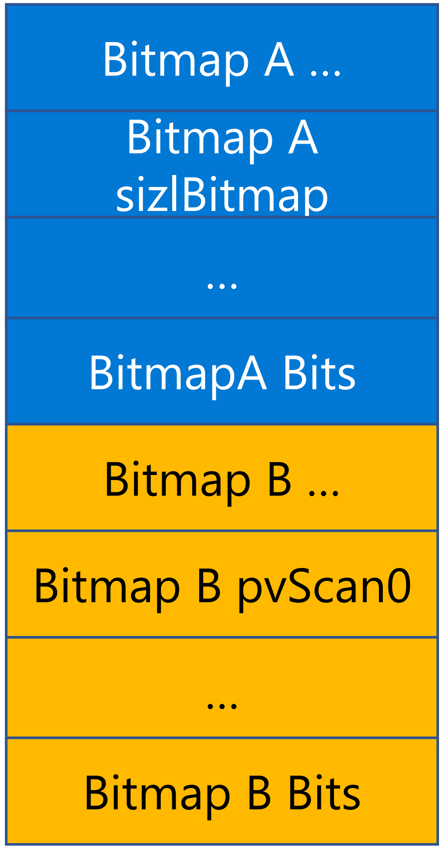
lpBits [in]

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

writes up to sizlBitmap data, into address pointed to by pvScan0.

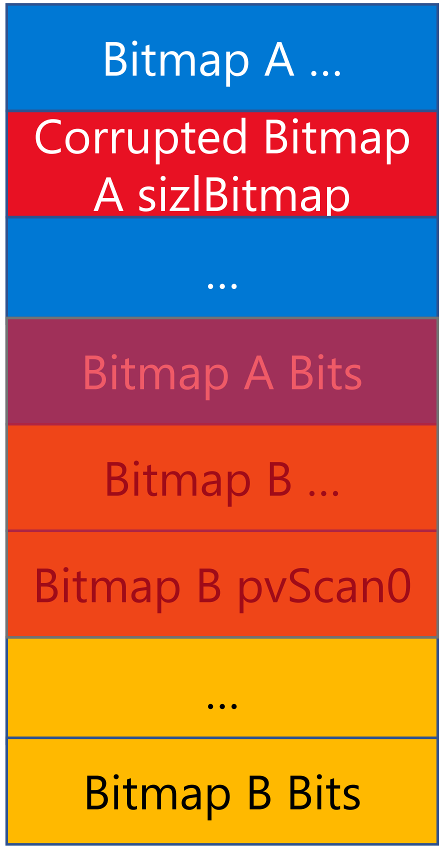
https://msdn.microsoft.com/en-us/library/windows/desktop/dd162962(v=vs.85).aspx

Abusing Bitmaps _SURFOBJ – Exploitation



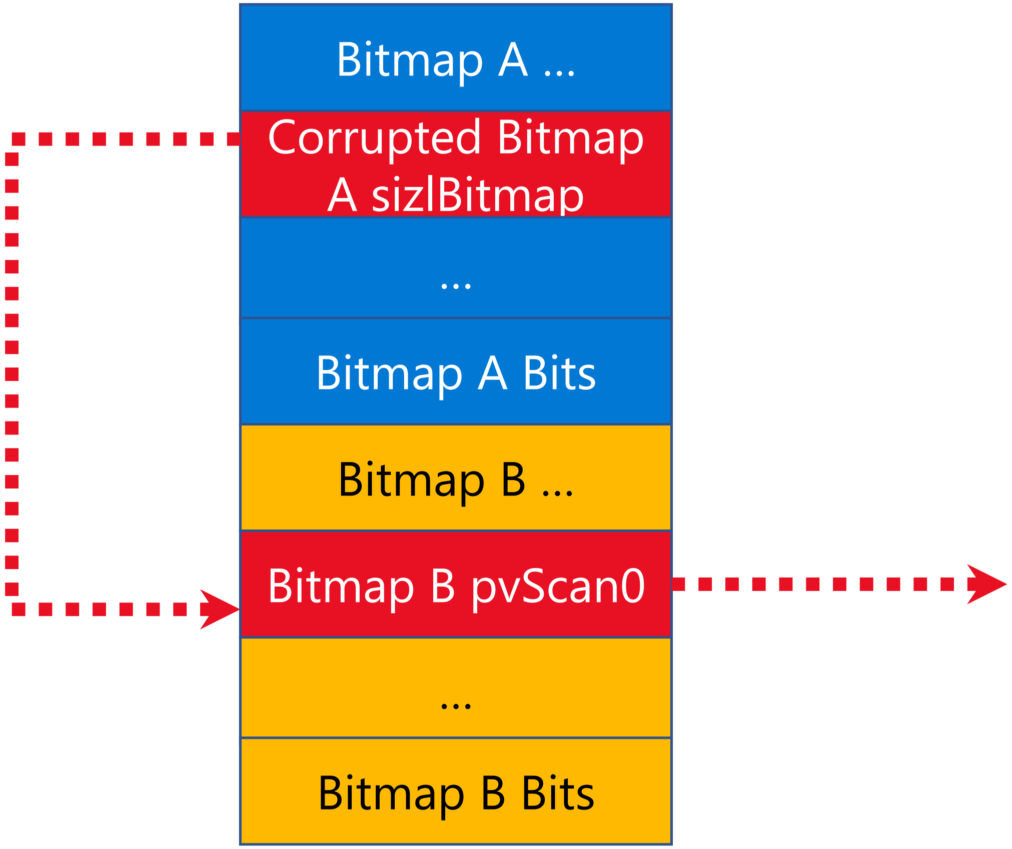
- Bitmap A & Bitmap B are two adjacent bitmaps that can read/write only their bits.

Abusing Bitmaps _SURFOBJ – Exploitation



- Use a kernel bug to corrupt Bitmap A sizlBitmap member.
- This will extend the Bitmap A size, gaining memory read/write relative to BitmapA.pvScan0 into the adjacent Bitmap B.
- Bitmap A will be the manager object that will be used to set the pointer to be read/write from.

Abusing Bitmaps _SURFOBJ – Exploitation



- Bitmap B will be the worker object.
- Use Bitmap A relative r/w to overwrite (set) Bitmap B pvScan0 to any location in kernel memory.
- Using Bitmap B read/write functions, allows arbitrary kernel memory read/write.

Read/write to/from any arbitrary kernel memory location pointed to by Bitmap B pvScan0

Abusing Palettes _PALETTE

Disclosed by Saif ElSherei @Saif_Sherei at Defcon 25 (2017)

Microsoft

Abusing Palettes _PALETTE

Object type _PALETTE

```
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             pfnGetNearest;  // 0x3c
    PFN             pfnGetMatch;    // 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             pfnGetNearest;  // 0x60
    PFN             pfnGetMatch;    // 0x68
    ULONGLONG       ullRGBTime;     // 0x70
    PRGB555XL       pRGBXlate;      // 0x78
    PALETTEENTRY    *pFirstColor;   // 0x80
    struct _PALETTE *ppalThis;      // 0x88
    PALETTEENTRY    apalColors[1];  // 0x90
} PALETTE64, *PPALETTE64;
```

Abusing Palettes _PALETTE - Allocation

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

Parameters

lplgpl [in]

A pointer to a **L**

palette.

```
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 palett

palPalEntry

Specifies an array of **PALETTEENTRY** struc

in the logical palette.

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

Members

pe

Allocate 2000 Palettes

```
HPALETTE hps;
LOGPALETTE *lPalette;
lPalette = (LOGPALETTE*)malloc(sizeof(LOGPALETTE)
+ (0x1E3 - 1) * sizeof(PALETTEENTRY));
lPalette->palNumEntries = 0x1E3;
lPalette->palVersion = 0x0300;
hps = CreatePalette(lPalette);
```

https://msdn.microsoft.com/en-us/library/windows/desktop/dd183507(v=vs.85).aspx

Abusing Palettes _PALETTE - Free

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

Parameters

hObject [in]

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

https://msdn.microsoft.com/en-us/library/windows/desktop/dd183539(v=vs.85).aspx

Abusing Palettes _PALETTE – Read Data

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

Parameters

```
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
```

must contain at least as many structures as specified by the *nEntries* parameter.

Reads up to nEntries from Index from data at address pointed to by pFirstColor

https://msdn.microsoft.com/en-us/library/windows/desktop/dd144907(v=vs.85).aspx

Abusing Palettes _PALETTE – Write Data

```
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
);
```

```
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
```

Write up to nEntries from index of data into address pointed to by pFirstColor

https://msdn.microsoft.com/en-us/library/windows/desktop/dd145077(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/dd183355(v=vs.85).aspx

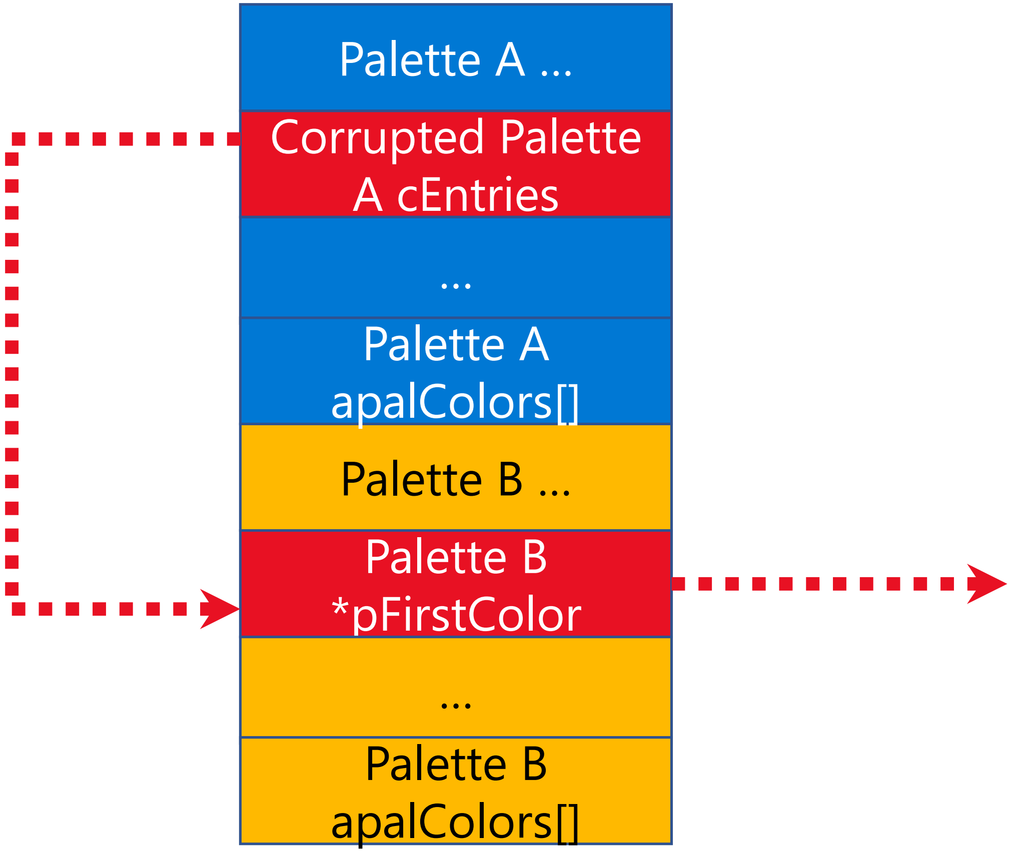
Abusing Palettes _PALETTE – Exploitation

- Palette A & B are two adjacent Palette objects that can read/write only their original entries.

Abusing Palettes _PALETTE – Exploitation

- Use Kernel exploit to corrupt Palette A cEntries, with a large value, expand its apalColors entries into the adjacent Palette B.
- Gaining kernel memory read/write relative to the location pointed to by Palette A pFirstColor member.
- Palette A will be the manager object, used to set the pointer to be read/write from.

Abusing Palettes _PALETTE – Exploitation



- Palette B will be the worker object.
- Use Palette A relative r/w to overwrite (set) Palette B pFirstColor to any location in kernel memory.
- Using Palette B read/write functions, allows arbitrary kernel memory read/write.

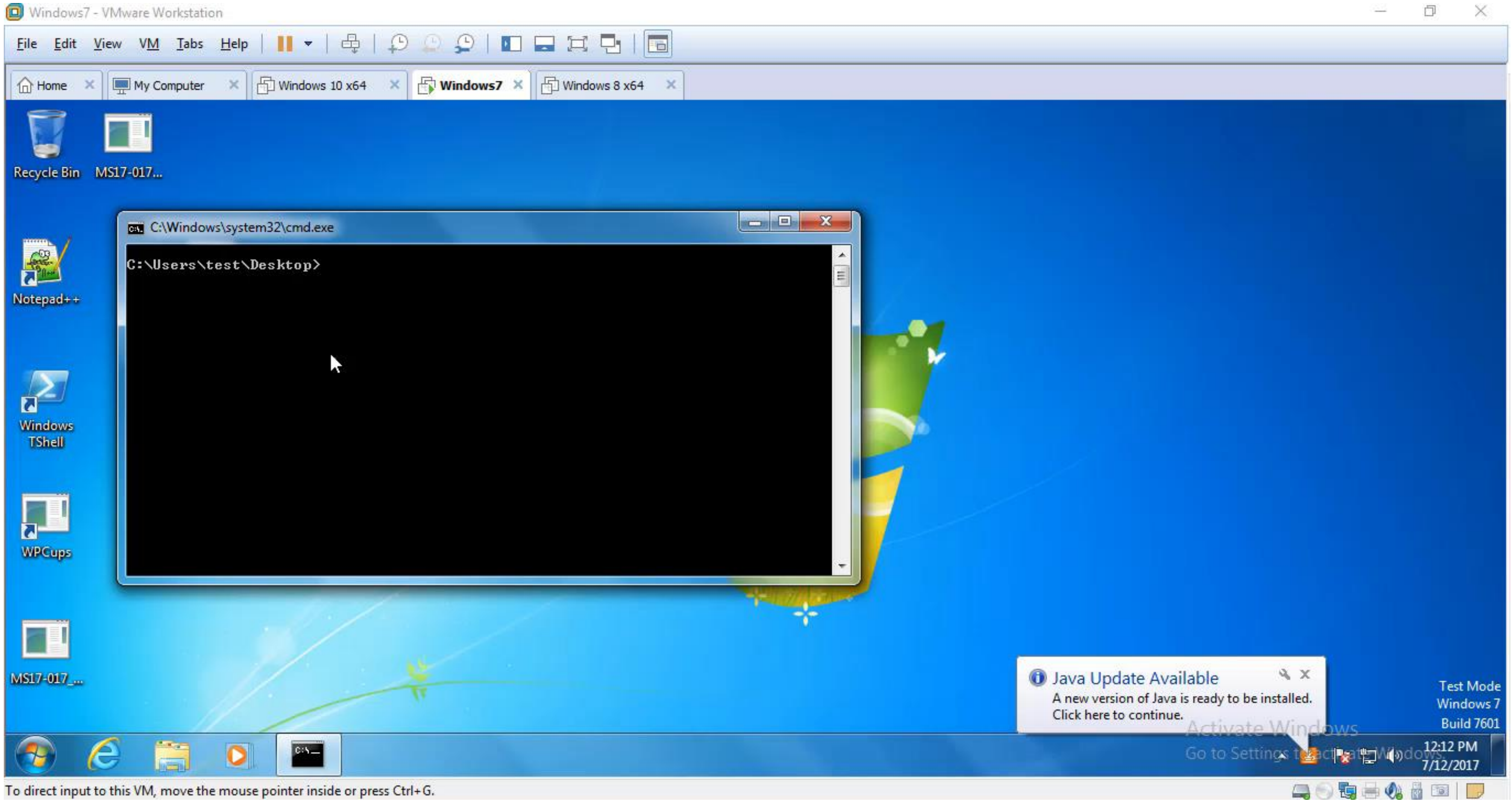
Read/write to/from any arbitrary kernel memory location pointed to by Palette B pFirstColor

Abusing Palettes _PALETTE – Restrictions

X86	X64

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

Demo



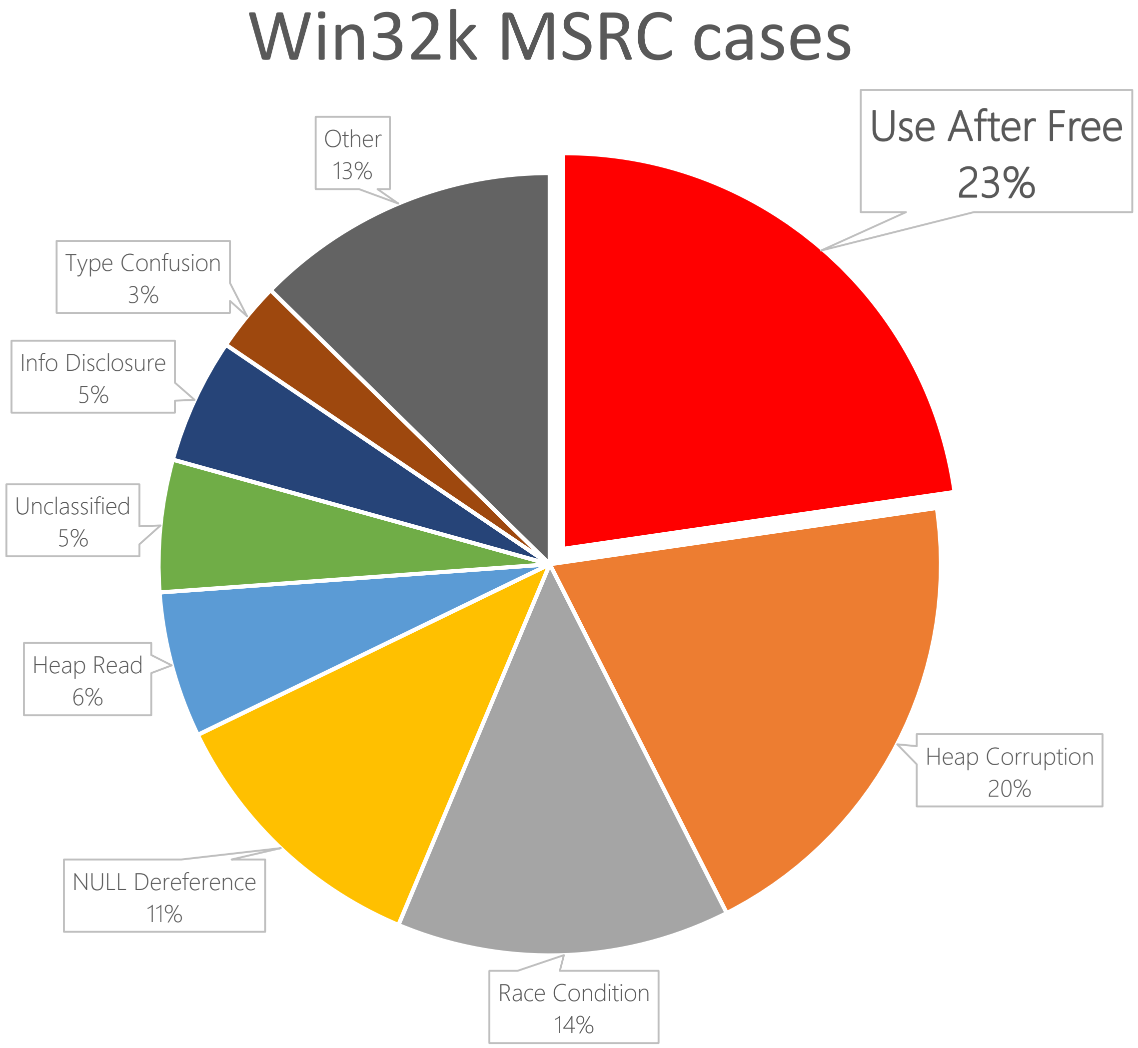
Windows7 - VMware Workstation
File Edit View VM Tabs Help
Home
My Computer
Windows 10 x64
Windows7
Windows 8 x64
Recycle Bin
MS17-017...
Notepad++
Windows TShell
WPCups
MS17-017_...
C:\Windows\system32\cmd.exe
C:\Users\test\Desktop>
Java Update Available
A new version of Java is ready to be installed.
Click here to continue.
Test Mode
Windows 7
Build 7601
12:12 PM
7/12/2017
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

The Death of Kernel Object Abuse (Mitigation)

Microsoft

The Type Isolation Mitigation

- We live in a world where there is a lot of buggy software, and a lot of crafty attackers.
- Unfortunately, we can’t fix every bug.
- What we need are mitigations: ways to make bugs more difficult, or even impossible, to exploit.
- We are raising the bar for hackers.



Our Threat Model

- We assume the attacker has found a UAF in one of the NTGDI or NTUSER types which we protect.
- They may cause this UAF to occur at arbitrary times.
- We assume the attacker does not have an arbitrary write – a UAF is a primitive you use to build an arbitrary write vulnerability.
- The attacker may have an arbitrary read vulnerability, though we’ve done a few things to make their lives harder if they don’t.

Not in Our Threat Model

- If you already have a write-what-where vulnerability, you’ve already won.
- We only protect a limited number of types, so exploiting a type we don’t protect is out of scope.

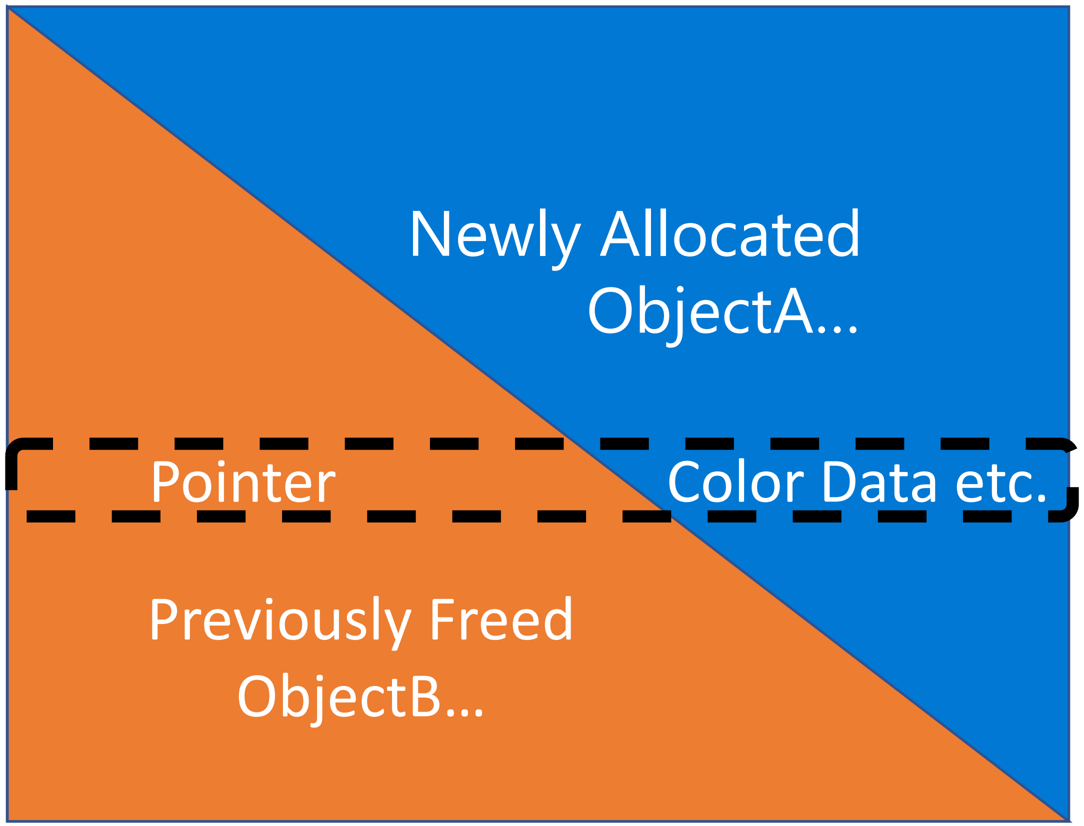
Type Isolation Doesn’t Prevent UAFs

- Type Isolation doesn’t actually stop UAFs, it just makes them very difficult to exploit.
- Since frees may happen at any time, it’s hard to detect them.
- To catch all UAFs, you need to check every pointer dereference, which is very slow.

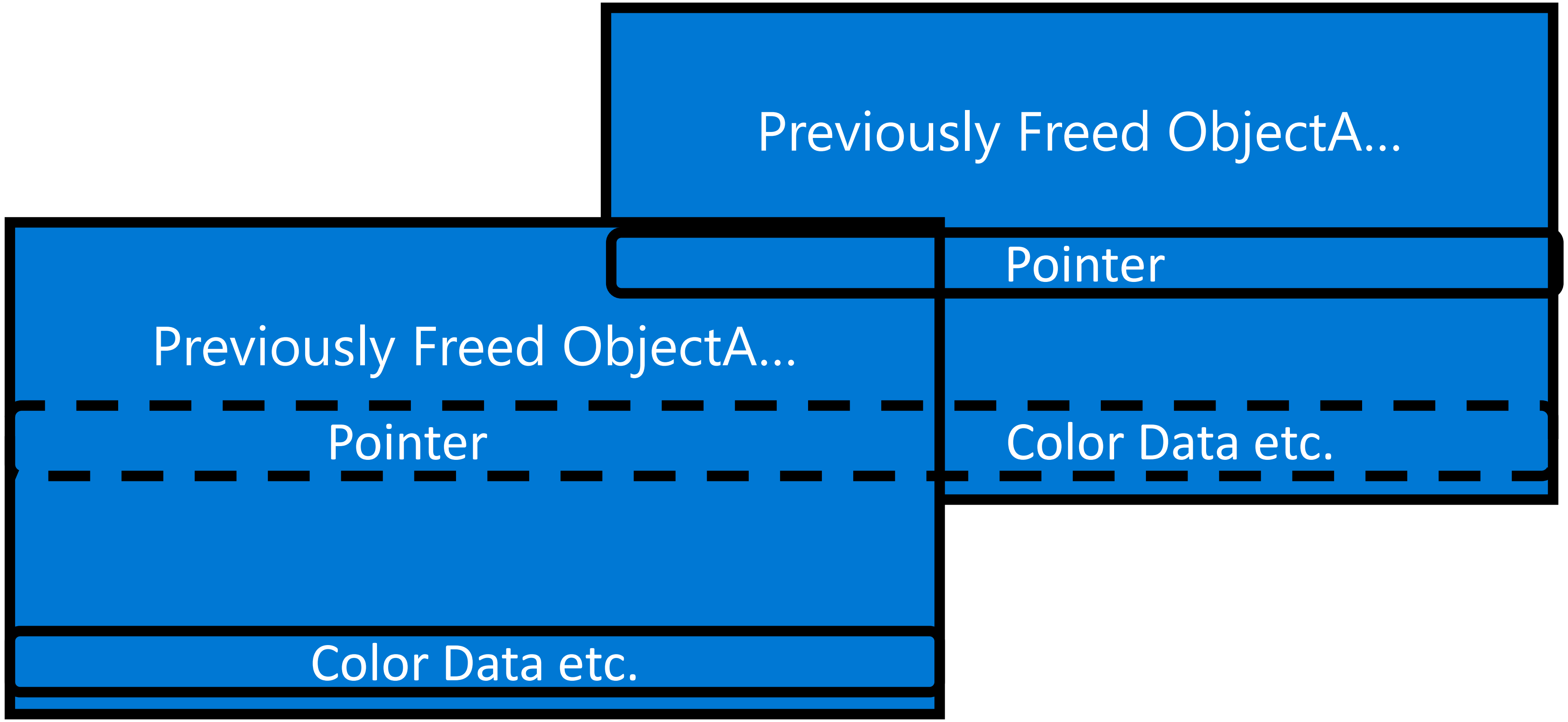
Deny the Attacker Control of Memory

- If an attacker can control the layout and contents of memory, they control the kernel.
- We change the layout of memory to be harder to exploit in the face of bugs, and deny the attacker control.

Overlapping Different Types of Objects

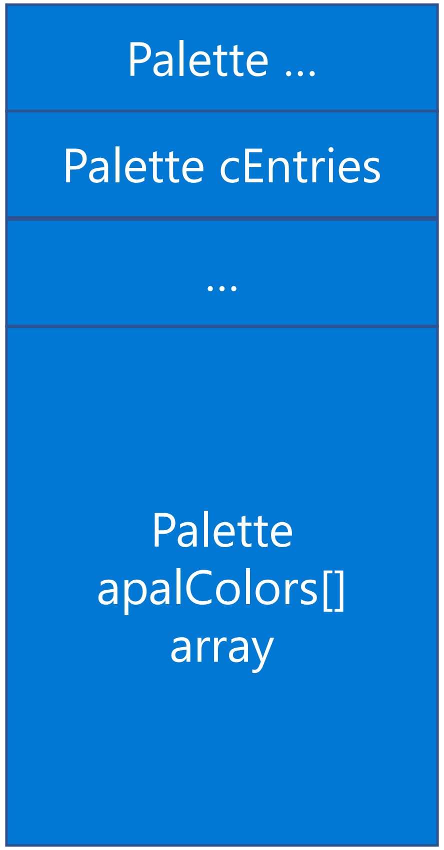


Overlapping the Same Types of Objects



How Type Isolation Works

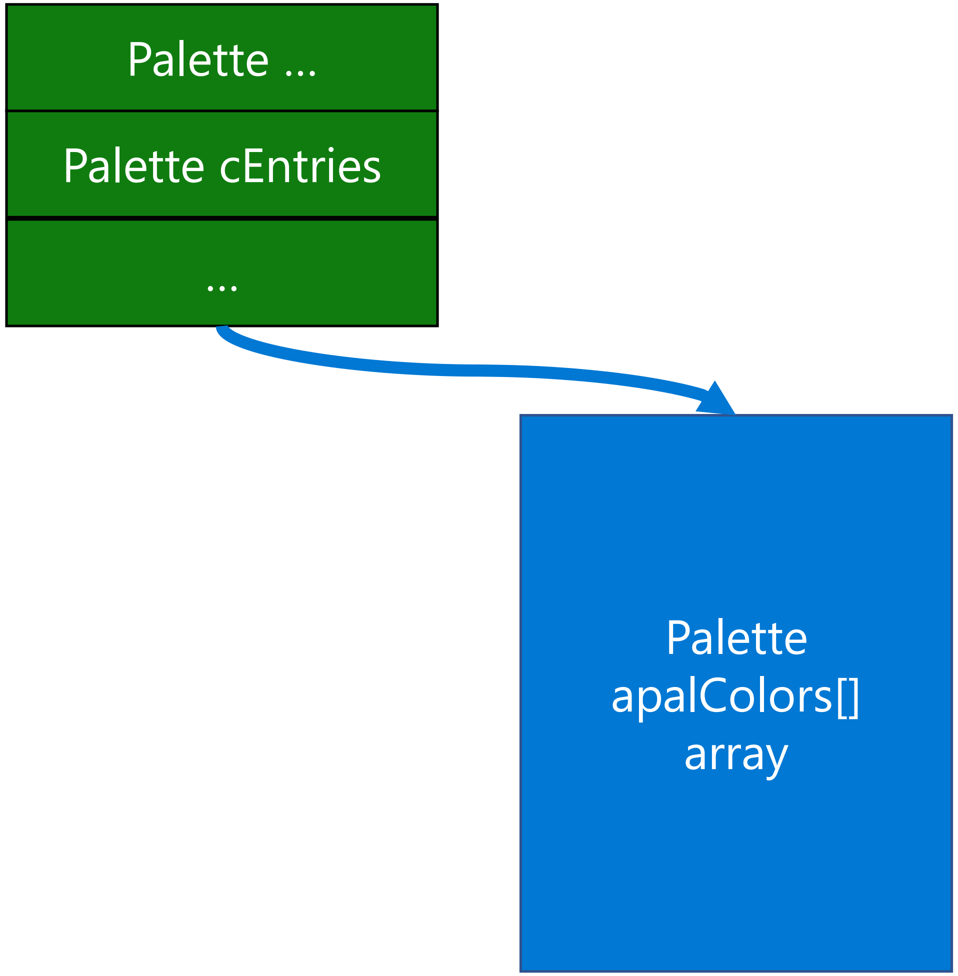
Before Type Isolation



Fixed sized green parts are in the isolated heap

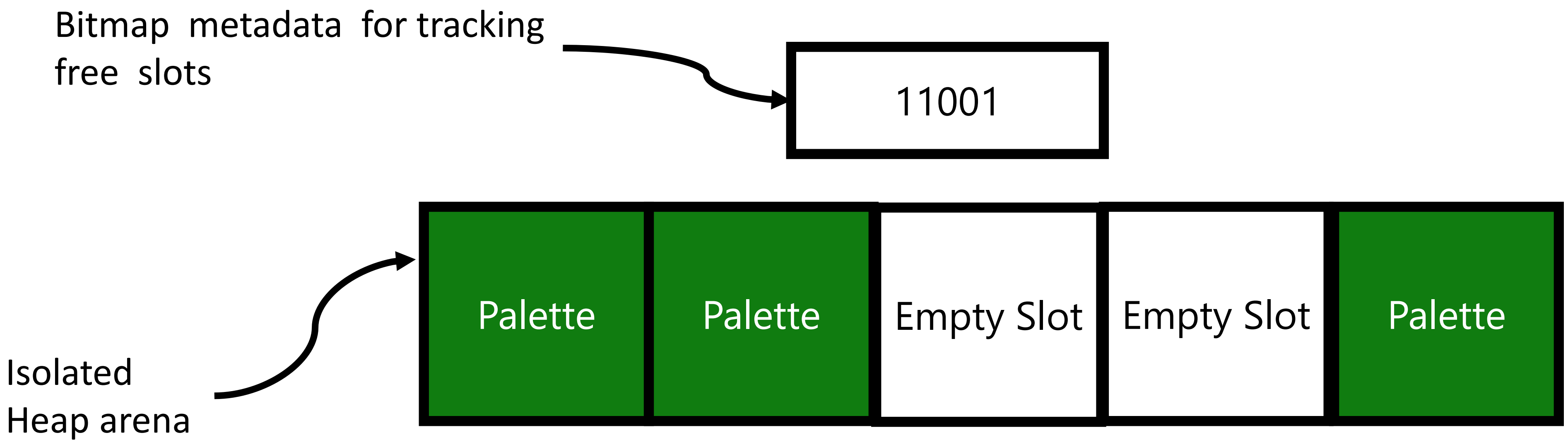
Variable sized blue parts are in the normal heap

After Type Isolation



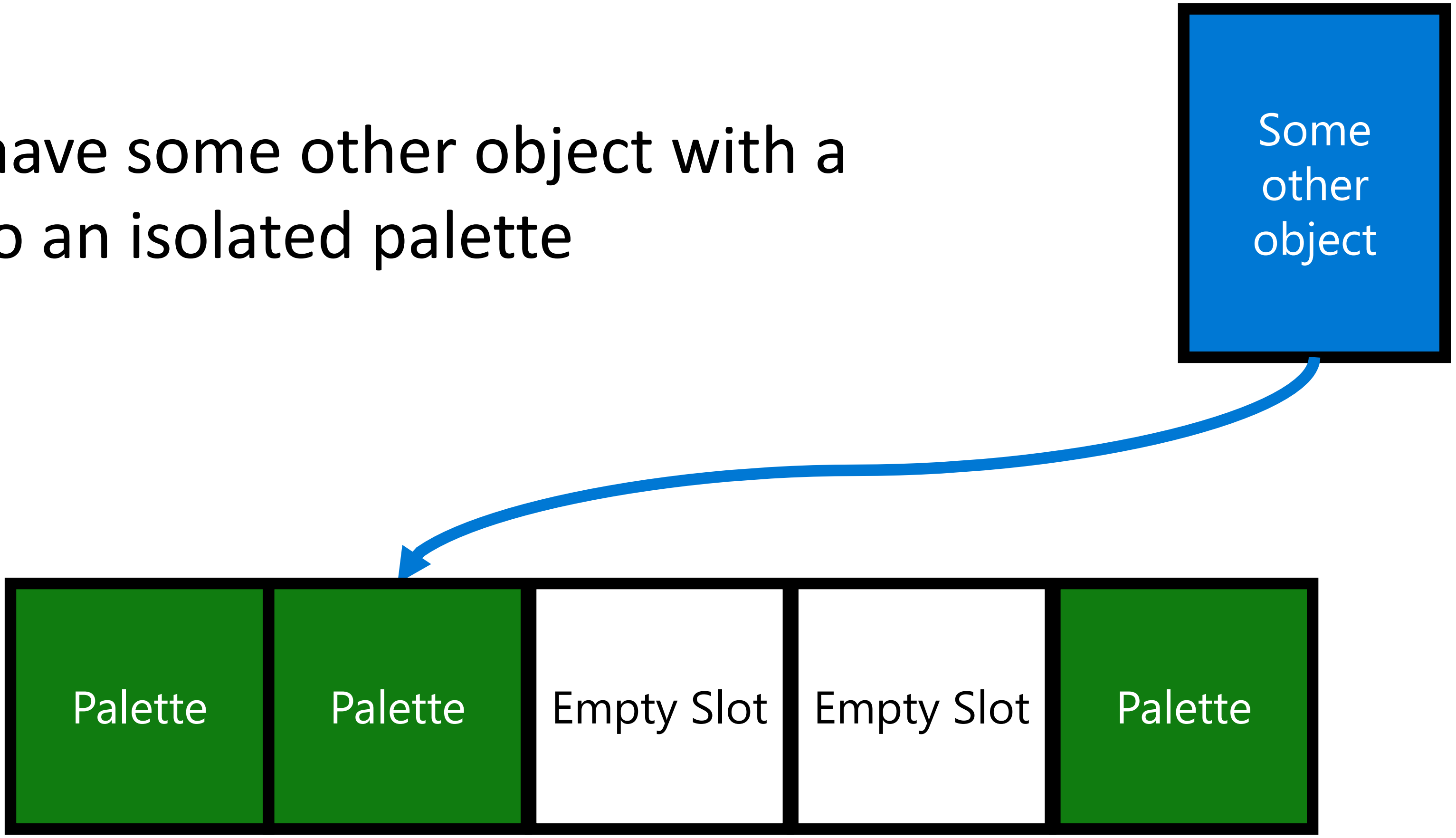
How Type Isolation Works

The isolated heap has a series of slots so two palettes can’t overlap. This way different types of fields like flags or sizes won’t overlap in the event of a UAF. Only palettes can be allocated from this heap.



UAF Scenario 1

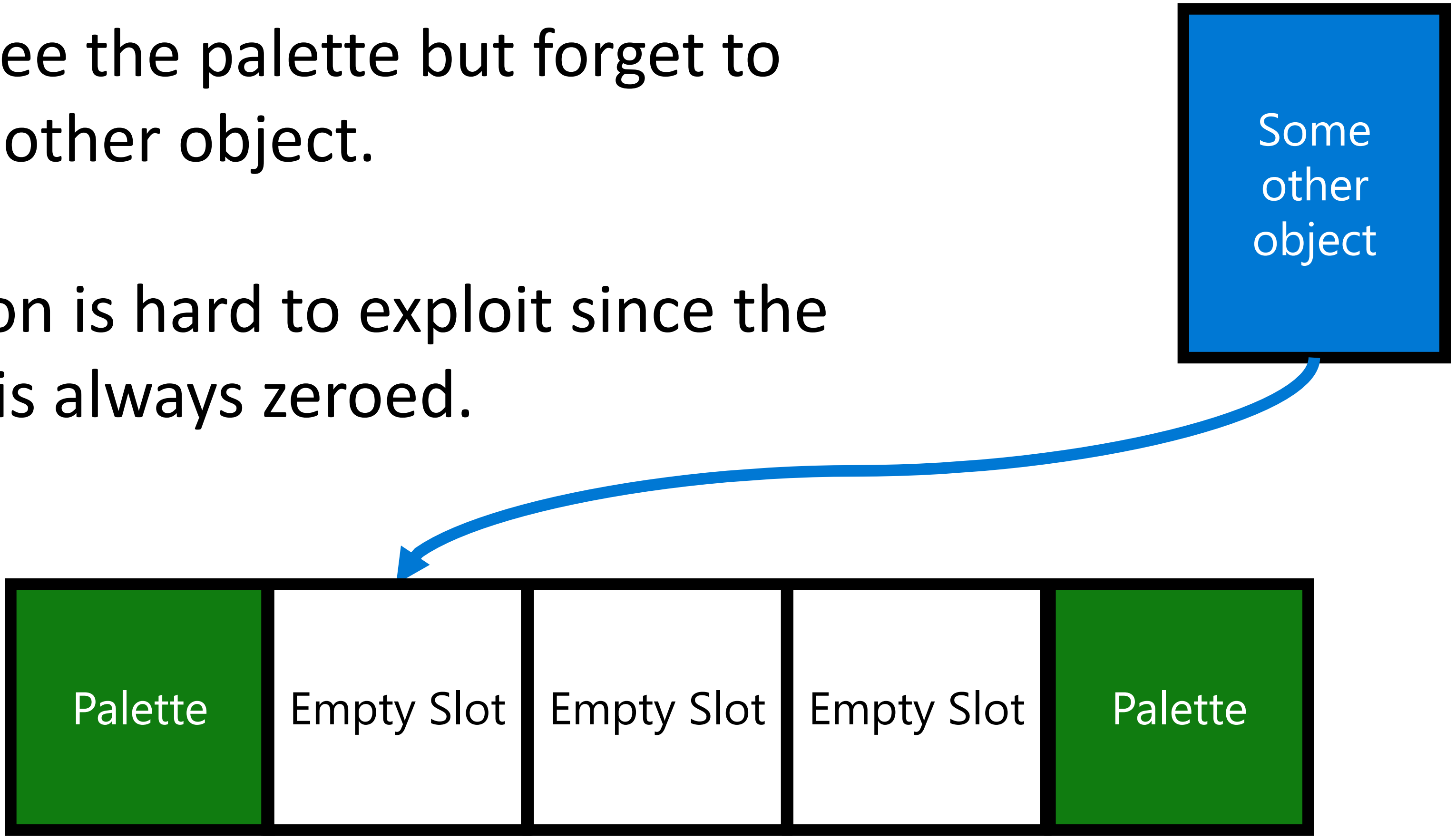
Say you have some other object with a pointer to an isolated palette



UAF Scenario 1

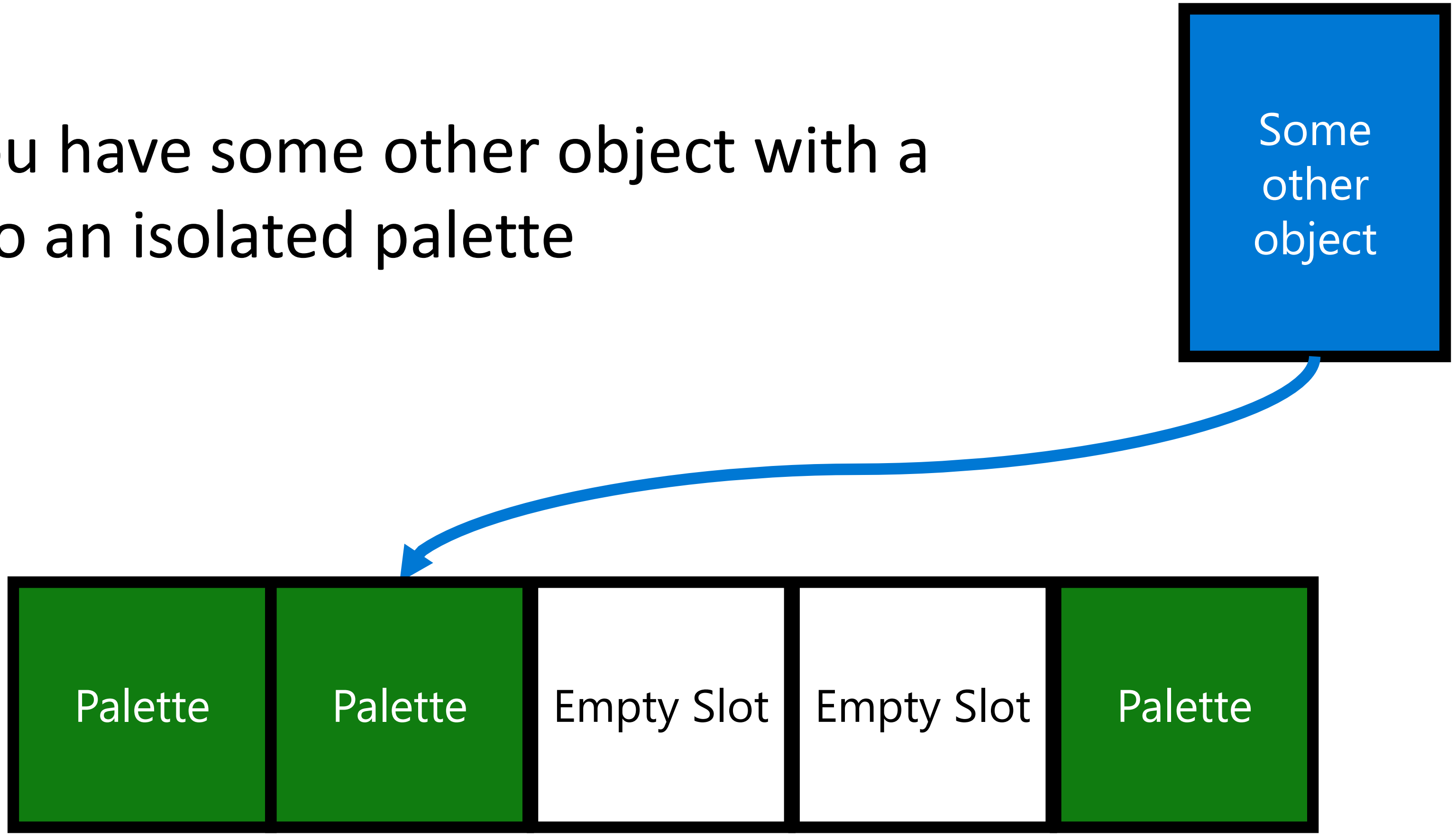
Then you free the palette but forget to update the other object.

This situation is hard to exploit since the empty slot is always zeroed.



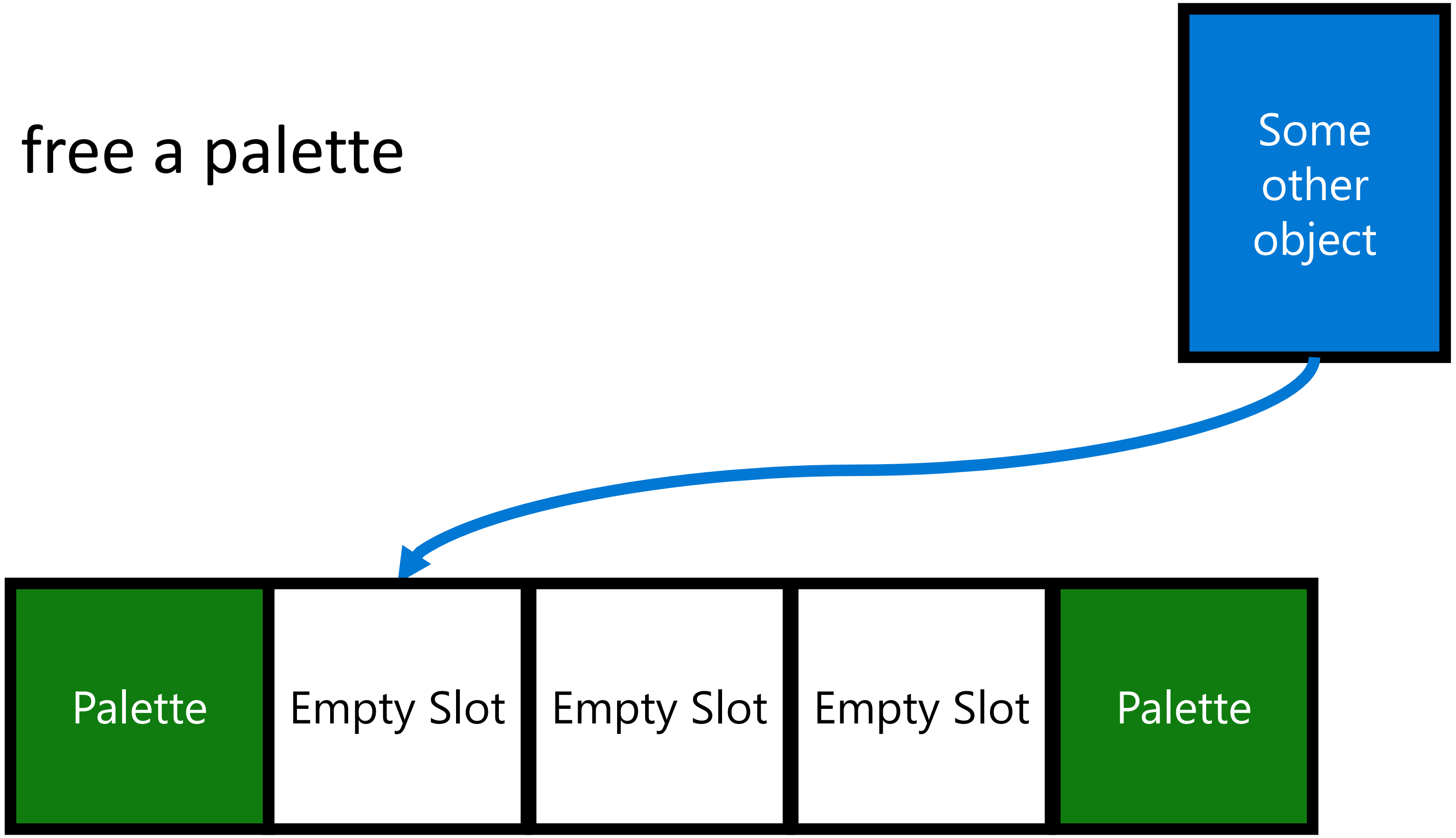
UAF Scenario 2

Again, you have some other object with a pointer to an isolated palette



UAF Scenario 2

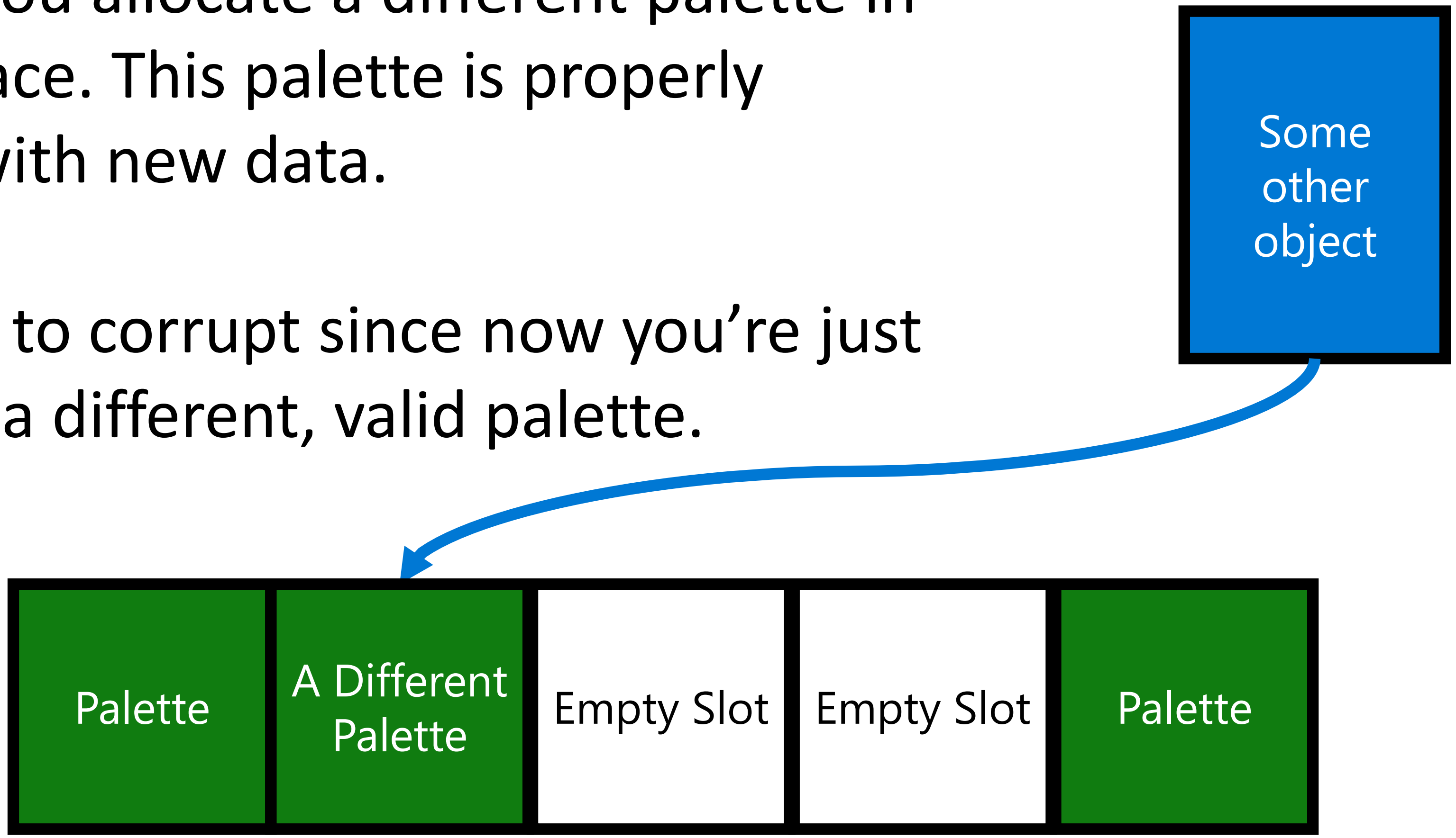
Then you free a palette



UAF Scenario 2

This time, you allocate a different palette in its same place. This palette is properly initialized with new data.

This is hard to corrupt since now you're just pointing to a different, valid palette.



Similar Work

- Adobe Flash introduced “Heap Partitioning” in 2015
- IE had IsoHeap, prior to adding a native code garbage collector
- Webkit added a similar feature which landed shortly after we did

Our Impact

“This definitely eliminates the commodity exploitation technique of using Bitmaps as targets for limited memory corruption vulnerabilities[.]”

~ Francisco Falcon of Quarkslabs talking about its impact on the SURFACE type alone

https://blog.quarkslab.com/reverse-engineering-the-win32k-type-isolation-mitigation.html

Q & A
Thanks

Microsoft