GDI魔术：漏洞利用中的利器

nEINEI/DiDi Labs

FIT · FREEBUF

WHO AM I

（aka：nEINEI）

安全研究工作@滴滴美国研究院

曾就职于McAfee,AVG

关注的领域：

恶意代码检测&漏洞攻防

高级威胁研究

也曾在Xcon,CanSecWest,AVAR,BlackHat 等国际安全会议发表过议题演讲。

目录

GDI: Graphics Device Interface

1 微软为应用程序提供图形设备接无关的一组API。

2 Nt早期版本将窗口管理和图形系统运行在用户地址空间,导致性能问题，从NT4.0 开始将窗口和图形移到内核态。

窗口管理:
窗口类，窗口，窗口钩子，窗口属性，消息，菜单，标题栏，滚动条，光标，虚拟键，剪切板…
由user32.dll模块提供接口

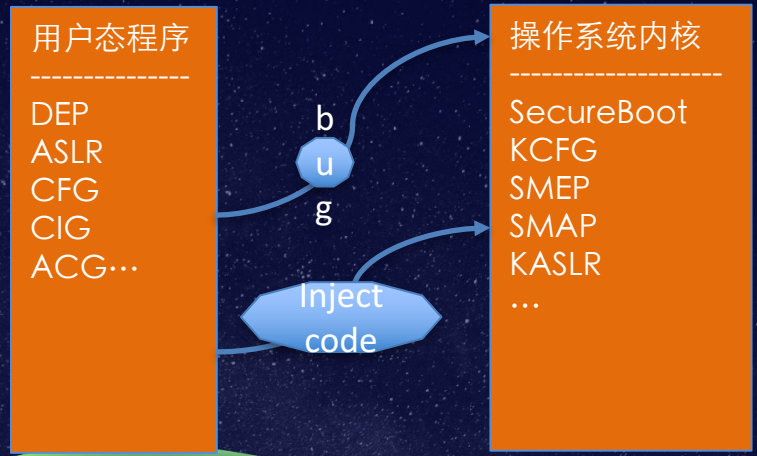
GDI,DirectDraw,D3D,OpenGl:
位图，画刷，剪裁，字体，直线，元文件，画笔，打印，矩形…

通过gdi32.dll模块提供接口

Win32k.sys (win32kfull.sys)

一 内核当中Read／Write Primitives的构造

当前要面对的各种操作系统的安全缓解机制



GDI:Graphics Device Interface

架起ring3 -> ring0 的漏洞利用通道

针对GDI对象构造出Read/Write Primitives

《Windows Kernel Exploitation : This Time Font hunt you down in 4 bytes》 KEEN Team，2015

这篇重要的paper里面提出了如何在优雅并且稳定的控制任意内核地址的数据的方法

CVE-2015-2387是对该技巧完美利用

Hacking Team事件中泄漏的：

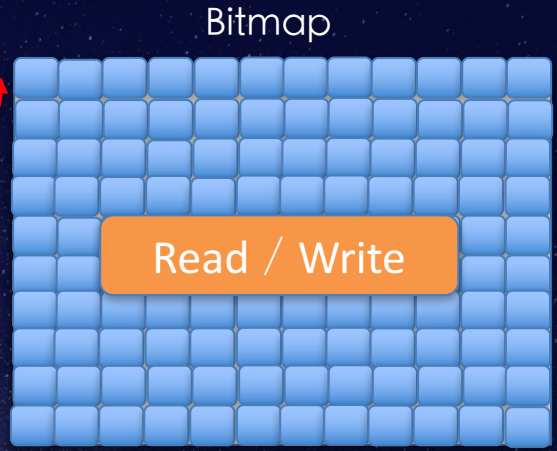
CVE-2015-2387:ATM 字体驱动程序中的漏洞可能允许特权提升 (3077657)
https://technet.microsoft.com/zh-cn/library/security/ms15-077.aspx

可能是第一个公开的ITW 0day 中使用的bitmap技巧的例子

Bitmap的数据结构

用户态API:

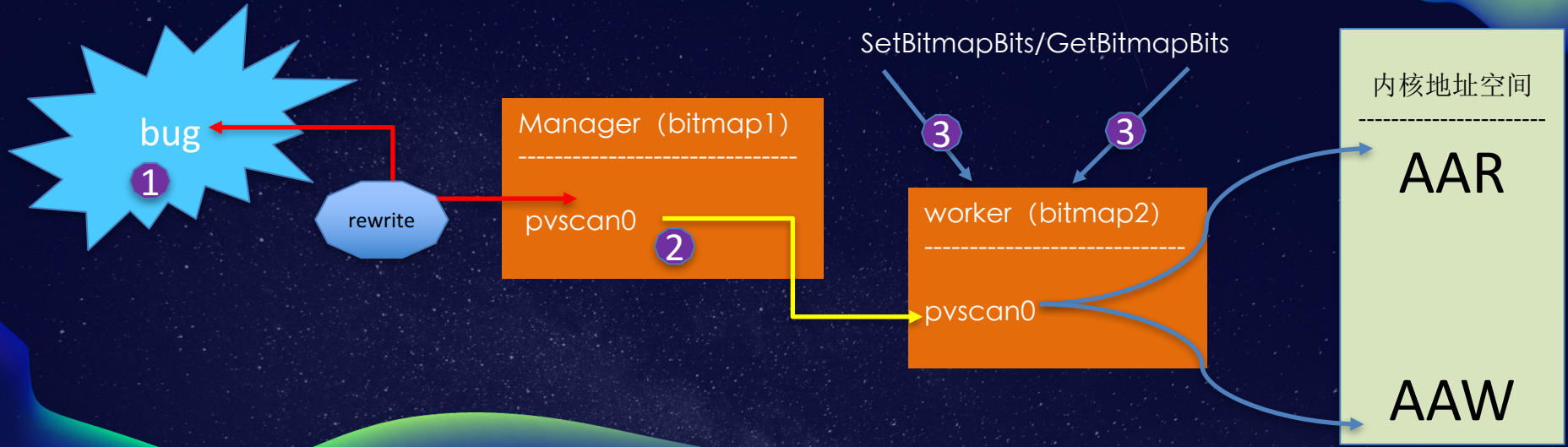
```
HBITMAP CreateBitmap(
 _In_     int  nWidth,
_In_      int  nHeight,
_In_      UINT cPlanes,
_In_      UINT cBitsPerPel,
 _In_ const VOID *lpvBits);
```



内核态对象：

```
typedef struct _SURFOBJ64 {
        ULONG64  dhsurf;
        ...
        SIZEL  sizlBitmap;
        ULONG64  cjBits;
        ULONG64  pvBits;
        ULONG64  pvScan0;
        LONG32  lDelta;
         ...;
}SURFOBJ64;
```

内核当中Read / Write Primitives的构造



如何获得GDI对象在内核当中的地址及pvscan0？

步骤：

1 获取PEB->GdiSharedHandleTable表

2 根据创建的GDI的handle计算出在该表中的偏移
3 offset = （handle&0xffff）+sizeof（GDICELL64）

4 取得pKernelddress

5 取得pvscan0 = pKernelddress + 0x50

```
using namespace std;
typedef struct {
PVOID64 pKernelAddress;
        USHORT wProcessId; // 0x08
        USHORT wCount; // 0x0a
        USHORT wUpper; // 0x0c
        USHORT wType; // 0x0e
        PVOID64 pUserAddress; //
0x10} GDICELL64; // sizeof = 0x18
```

二 MS对Bitmap利用的缓解策略

Anniversary Update对Bitmap利用的缓解措施

Black Hat USA 2016 Microsoft Windows 10 Anniversary Update (v.1607)

《Windows 10 Mitigation Improvements》by David Weston/Matt Miller

1 Page table self-map and PFN database are randomized

2 SIDT/SGDT kernel address disclosure is prevented when Hyper-V is enabled

3 GDI shared handle table no longer discloses kernel addresses

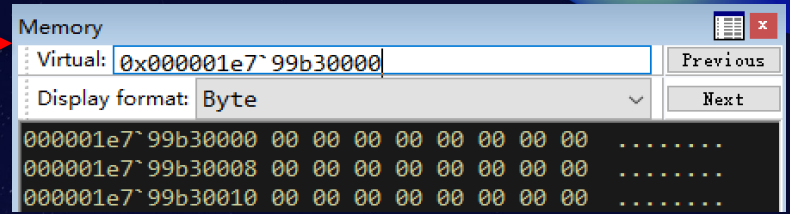
以上，通过进程PEB获得GDI Shared handle table的技巧失效

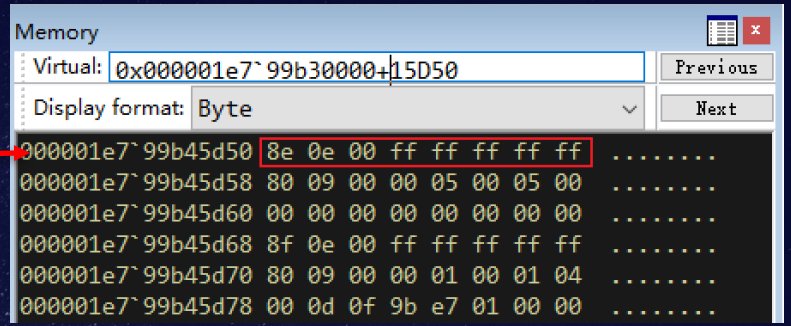
Mitigations： Windows 10 Creators update preview build 14986


```
0:005> !peb at 000000526dad3000

+0x0ec MaximumNumberOfHeaps : 0x10
+0x0f0 ProcessHeaps     : 0x00007ffc`e7760b40  -> 0x000001e7`998a0000 Void
+0x0f8 GdiSharedHandleTable : 0x000001e7`99b30000 Void
 +0x100 ProcessStarterHelper : (null)
 +0x108 GdiDCAttributeList : 0x14

hBitmap = 0x50e8e =>
Offset = (0x50e8e & 0xffff ) * sizeof (GDICELL64) = 0x15D50
pKernelAddress = GdiSharedHandleTable  + 0x15D50 = 》
```

000001e7`99b45d50

FAILED





三 从tagWnd的构造谈起

什么时候开始关注对窗口对象的利用技术？

《Kernel Attacks through User-Mode Callbacks》Tarjei Mandt,Blackhat usa2011.

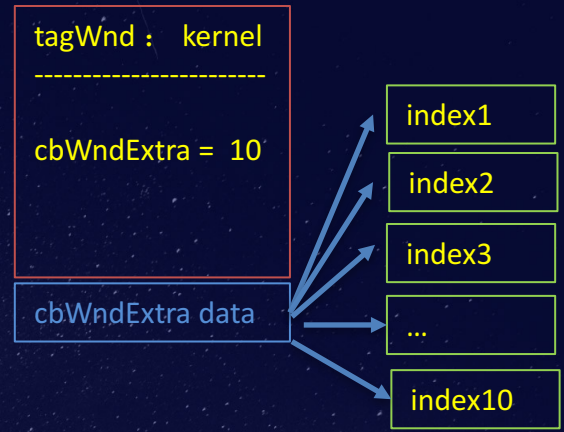
《Duqu2.0 Win32k Exploit Analysis》by Jeong Wook oh,VB2015 conf.

tagWnd里面的关键结构

```
WNDCLASSEX wnd = { 0x0 };
...
wnd.cbWndExtra = 0x10;
int result = RegisterClassEx(&wnd);

//创建一个windows
CreateWindowEx(0,
wnd.lpszClassName,
 NULL, 20, CW_USEDEFAULT,
CW_USEDEFAULT,
 CW_USEDEFAULT,
 CW_USEDEFAULT, NULL, NULL, NULL, NULL);
```

bug

```
kd> dt tagWND
win32k!tagWND
   +0x000 head
   +0x014 state
   ...
   +0x014 bDestroyed
   +0x018 state2
   ...
   +0x020 bWS_BORDER
   +0x020 bMaximized
   +0x020 bWS_CLIPCHILDREN
   +0x020 bWS_CLIPSIBLINGS
   +0x020 bDisabled
   +0x020 bVisible
   +0x020 bMinimized
   +0x020 bWS_CHILD
   +0x020 bWS_POPUP
   +0x024 hModule
   +0x074 spmenuSys
   +0x078 spmenu
   +0x07c hrgnClip
   +0x080 hrgnNewFrame
   +0x084 strName
   +0x090 cbwndExtra
   +0x094 spwndLastActive
```



构造Read/Write Primitives通过tagWnd方法

```
BOOL WINAPI SetWindowText( _In_     HWND    hWnd,
_In_opt_ LPCTSTR lpString );

int WINAPI GetWindowText( _In_  HWND   hWnd,
_Out_ LPTSTR lpString,
_In_  int    nMaxCount );
```

InternalGetWindowText --> Read
NtUserDefSetText - > Write

```
kd> dt tagWND
win32k!tagWND
…
+0x080 hrgnNewFrame
+0x084 strName
+0x090 cbwndExtra
+0x094 spwndLastActive …
+0x0ac bInSendString
```

修改窗口的strName指针再次获得Read/Write Primitives

四 MS对tagWnd利用的缓解策略

MS case by case的解决方案

《Hardening Windows 10 with zero-day exploit mitigations》 by mmpc，January 13,2017

https://blogs.technet.microsoft.com/mmpc/2017/01/13/hardening-windows-10-with-zero-day-exploit-mitigations/016

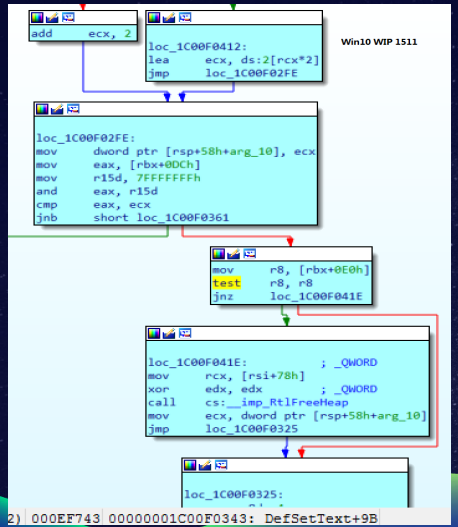
增加了一个校验函数DesktopVerifyHeapLargeUnicodeString

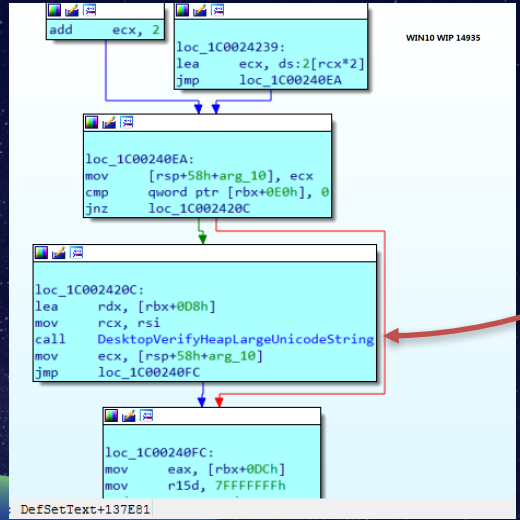
对Get/SetWindowText的输入buffer进行进行校验：

```
 # Child-SP          RetAddr           Call Site
00 ffff8b00`65a92068 fffff800`36a5c96a nt!DbgBreakPointWithStatus
01 ffff8b00`65a92070 fffff800`36a5c359 nt!KiBugCheckDebugBreak+0x12
02 ffff8b00`65a920d0 fffff800`369d3094 nt!KeBugCheck2+0x8a5
03 ffff8b00`65a927e0 ffffdeb2`f731c1fe nt!KeBugCheckEx+0x104
04 ffff8b00`65a92820 ffffdeb2`f71e4f96 win32kfull!DesktopVerifyHeapPointer+0x137252
05 (Inline Function) --------`-------- win32kfull!DesktopVerifyHeapRange+0x15
06 ffff8b00`65a92860 ffffdeb2`f71e421b win32kfull!DesktopVerifyHeapLargeUnicodeString(struct ta
07 ffff8b00`65a928a0 ffffdeb2`f720c99c win32kfull!DefSetText(struct tagWND * pwnd = 0xffffded1`
08 ffff8b00`65a92900 ffffdeb2`f720c50a win32kfull!xxxRealDefWindowProc(struct tagWND * pwnd = 0
09 ffff8b00`65a92a80 ffffdeb2`f71e51ec win32kfull!xxxWrapRealDefWindowProc(struct tagWND * pwnd
```

Windows 10 Anniversary update

增加了一个校验函数DesktopVerifyHeapLargeUnicodeString





Patched ,增加了一个校验函数：

DesktopVerifyheapLargeUnicodeString

不能再用Get/SetWindowText获得RW Primitives

```
ULONG_PTR __fastcall DesktopVerifyHeapLargeUnicodeString(__int64 a1, ULONG_PTR a2)
{
  __int64 v2; // rbx@1
  ULONG_PTR v3; // r8@1
  int v4; // er9@2
  ULONG_PTR v5; // rdx@4
  __int64 v6; // r9@5
  ULONG_PTR v7; // rdi@5

  v2 = a1;
  v3 = a2;
  if ( *(_DWORD *)a2 & 1
    || (v4 = *(_DWORD *)(a2 + 4), v4 & 1)
    || *(_DWORD *)a2 >= (v4 & 0x7FFFFFFFu)
    || (v5 = *(_QWORD *)(a2 + 8), v5 & 0xF) )
  {
    KeBugCheckEx(0x164u, 7ui64, v3, *(_QWORD *)(a1 + 120), *(_DWORD *)(a1 + 128));
  }
  v6 = v4 & 0x7FFFFFFF;
  v7 = v5 + v6;
  if ( v5 + v6 < v5 )
    KeBugCheckEx(0x164u, 6ui64, v5, *(_QWORD *)(a1 + 120), *(_DWORD *)(a1 + 128));
  DesktopVerifyHeapPointer(a1, v5);
  return DesktopVerifyHeapPointer(v2, v7 - 1);
```

对目标窗口的strName指针进行判断，不符合合法指针范围，内存对齐，超过分配buffer空间的都将引发，KeBugCheckEx。

真的失去了利用GDI对内核操纵吗？

五 复活Bitmap的利用技巧

获得Bitmap对象内核地址

《Abusing GDI for ring0 exploit primitives： Reloaded》 by CoreSecurity,ekoparty 2016

这篇papaer里面，着重讨论了如何利用堆风水的方式，来获得bitmap对象的地址。

再次利用Fenshui获得Bitmap对象内核地址

1 构造可以预测kaddress地址的user object(利用**user32!gSharedInfo**)

a）object size < 4k , NtUserConvertMemHandle/NtUserSetClipboardData

b） object size > 4k，CreateAcceleratorTableA/DestroyAcceleratorTable

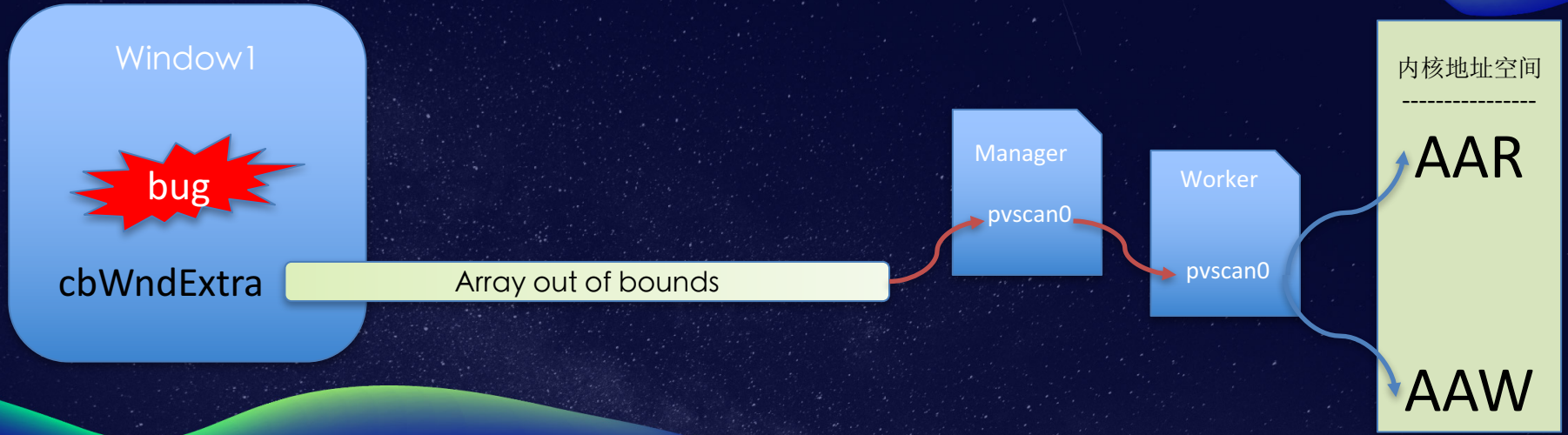
2 释放掉这些user object

3 放入Bitmap并获得对应的内核对象地址

六 翻转Bitmap的利用

抛砖引玉，分享个思路。

之前，我们要控制指针(QWORD/DWORD/WORD/Byte)



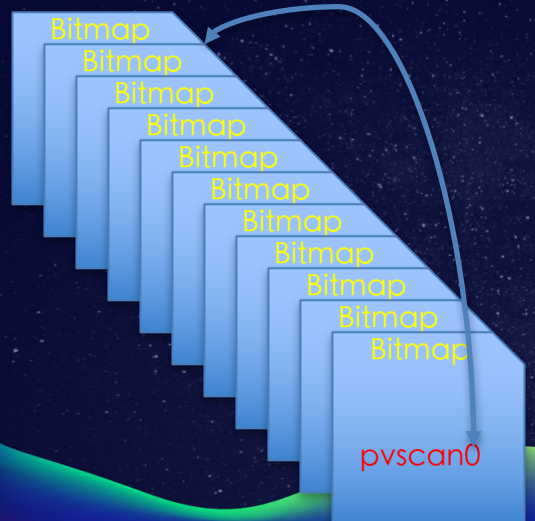
如何不完全控制也能做到RW Primitives？

1 创建 > 100 bitmap 对象
2 依次尝试使用width * high = 5k， 20k ， 80k，100k 不同大小的 bitmap

3 如果一个bitmap对象pvscan0和另外一个bitmap对象的pvscan0存在相关连的地方我们就可以利用

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

通常需要完全控制改写pvscan0的指针




```
Select C:\Users\jf\Desktop\TestingFBTB.exe
HBITMAP of surface's pvscan0 of address[89]:fffff90142ffb050
HBITMAP of surface's pvscan0 of address[90]:fffff90143005050
HBITMAP of surface's pvscan0 of address[91]:fffff9014300d050
HBITMAP of surface's pvscan0 of address[92]:fffff90143017050
HBITMAP of surface's pvscan0 of address[93]:fffff9014301f050
HBITMAP of surface's pvscan0 of address[94]:fffff90143029050
HBITMAP of surface's pvscan0 of address[95]:fffff90143031050
HBITMAP of surface's pvscan0 of address[96]:fffff9014303b050
HBITMAP of surface's pvscan0 of address[97]:fffff90143043050
HBITMAP of surface's pvscan0 of address[98]:fffff9014304d050
HBITMAP of surface's pvscan0 of address[99]:fffff90143055050
first spary HBITMAP of address:fffff9014278a050
index:[35]->bit filp index:[41]spary HBITMAP of address:fffff90142cdc050
index:[49]->bit filp index:[56]spary HBITMAP of address:fffff90142d6f050
index:[59]->bit filp index:[66]spary HBITMAP of address:fffff90142dc9050
index:[61]->bit filp index:[68]spary HBITMAP of address:fffff90142ddb050
index:[63]->bit filp index:[70]spary HBITMAP of address:fffff90142ded050
index:[77]->bit filp index:[84]spary HBITMAP of address:fffff90142fcf050
index:[79]->bit filp index:[86]spary HBITMAP of address:fffff90142fe1050
index:[81]->bit filp index:[88]spary HBITMAP of address:fffff90142ff3050
index:[91]->bit filp index:[98]spary HBITMAP of address:fffff9014304d050
end,Please press any key to exit!
```

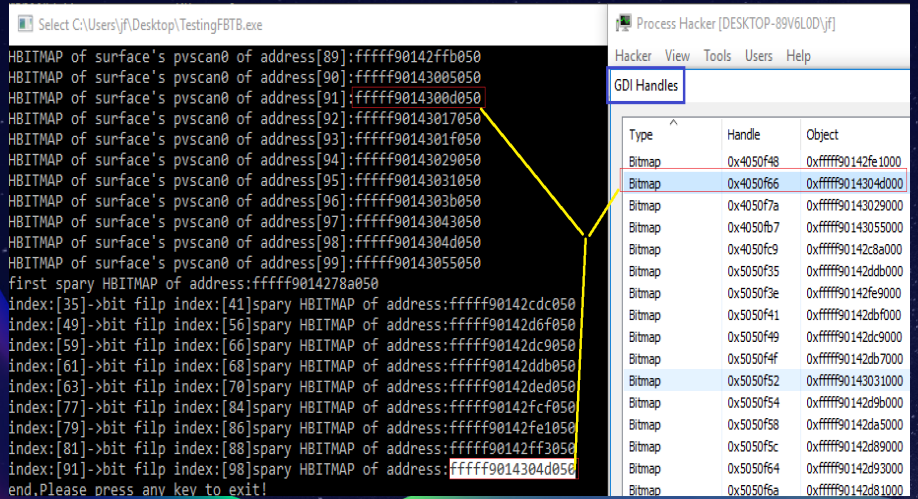
Process Hacker [DESKTOP-89V6L0D\jf]

Hacker View Tools Users Help

GDI Handles

Type	Handle	Object
Bitmap	0x4050f48	0xfffff90142fe1000
Bitmap	0x4050f66	0xfffff9014304d000
Bitmap	0x4050f7a	0xfffff90143029000
Bitmap	0x4050fb7	0xfffff90143055000
Bitmap	0x4050fc9	0xfffff90142c8a000
Bitmap	0x5050f35	0xfffff90142ddb000
Bitmap	0x5050f3e	0xfffff90142fe9000
Bitmap	0x5050f41	0xfffff90142dbf000
Bitmap	0x5050f49	0xfffff90142dc9000
Bitmap	0x5050f4f	0xfffff90142db7000
Bitmap	0x5050f52	0xfffff90143031000
Bitmap	0x5050f54	0xfffff90142d9b000
Bitmap	0x5050f58	0xfffff90142da5000
Bitmap	0x5050f5c	0xfffff90142d89000
Bitmap	0x5050f64	0xfffff90142d93000
Bitmap	0x5050f6a	0xfffff90142d81000

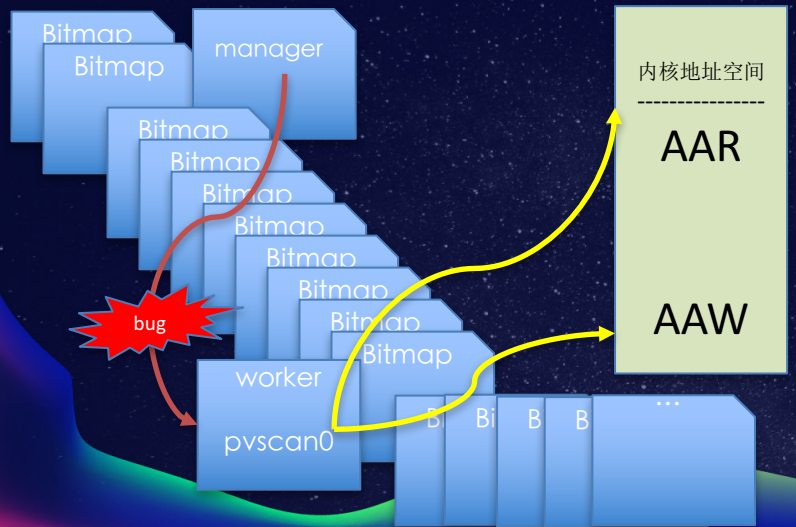
仅仅控制1个bit，可改写pvscan0的指针



Index[91] = fffff9014300d050 =… 0100 0011 0000 0000 1101

Index[98] = fffff9014304d050 = …0100 0011 0000 0100 1101

仅仅控制1个bit，可改写pvscan0的指针



Bitmap Index[91] = fffff9014300d050 =⋯ 0100 0011 0000 0000 1110

Bitmap Index[98] = fffff9014304d050 = ⋯ 0100 0011 0000 0100 1110

在GDI32/USER32模块中仍然有可挖掘的...

```
0:011> x user32!*set*
00007ffc`e36c1854 USER32!SetWindowRgn$fin$0 (void)00007ffc`e36b0350 USER32!ChangeDisplaySettingsExA
(void)00007ffc`e367d6f0 USER32!ClientThreadSetup (void)00007ffc`e3679850 USER32!SetWindowRgn
(void)00007ffc`e36877a4 USER32!ChangeDispSettingsNotificationW (void)00007ffc`e36907c0
USER32!SetWindowServicesDestroyCallback (void)00007ffc`e36b0420 USER32!ChangeDispSettingsNotificationA
(void)00007ffc`e36acb40 USER32!xxxSetFrameTitle (void)00007ffc`e3690f80
USER32!RegisterPowerSettingNotification (void)00007ffc`e36927f4 USER32!DlgSetFocus 、

0:011> x gdi32!*set*
00007ffc`e3156560 GDI32!bGetANSISetMap (void)00007ffc`e3157630 GDI32!SetDIBits (void)00007ffc`e31b4c40
GDI32!SetColorOptimization (void)00007ffc`e3158260 GDI32!SetBrushOrgEx (void)00007ffc`e31b0230
GDI32!SetMiterLimit (void)00007ffc`e31526a0 GDI32!GdiProcessSetup (void)00007ffc`e31997d0 GDI32!SetLayout
(void)00007ffc`e3144a10 GDI32!SetViewportOrgEx (void)00007ffc`e3153a80 GDI32!SetDCBrushColor
(void)00007ffc`e31b7db0 GDI32!AutoBuffer<unsigned char,160>::SetSize (void)00007ffc`e31b3c20
GDI32!MRSETVIEWPORTORGEX::bPlay (void)00007ffc`e31b07d0 GDI32!MRSETTEXTALIGN::bPlay
(void)00007ffc`e314f250 GDI32!SetTextAlign (void)
```

GDI的魔术一直在发生...

Palette构造RW Primitives
《Win32k Dark Composition：Attacking the Shadow Part of Graphic Subsystem》CanSecWest,2017
《Demystifying Kernel Exploitation by Abusing GDI Objects》DefCon,2017

利用Bitmap. sizlBitmap越界
利用tagCLS分配在内核，从新利用tagWND的RW Primitives.

《TAKING WINDOWS 10 KERNEL EXPLOITATION TO THE NEXT LEVEL – LEVERAGING WRITE-WHAT-WHERE VULNERABILITIES IN CREATORS UPDATE》Blackhat,2017

…

Thanks!

滴滴出行安全应急响应中心
Didichuxing Security Response Center

漏洞收集 | 安全说 | DDCTF

DSRC-滴滴出行安全应急响应中心，是安全研究者反馈滴滴出行产品和业务安全问题的官方平台。DSRC旨在加强滴滴出行与安全业界的合作，提升滴滴出行整体安全水平，打造健康安全的互联网出行生态。