A New CVE-2015-0057 Exploit Technology

wang yu

Black Hat - ASIA, 2016

Part One

Introduction

Introduction

- About me (yu.wang@fireeye.com)

- Background

It is worth noting that in 2015 alone, we have repeatedly caught APT class zero-day attacks - all of which target the Win32K subsystem's User Mode Callback mechanism. This leads us to re-visit this old-school kernel attack surface.

This talk will focus on CVE-2015-0057 and the User Mode Callback mechanism.

Introduction

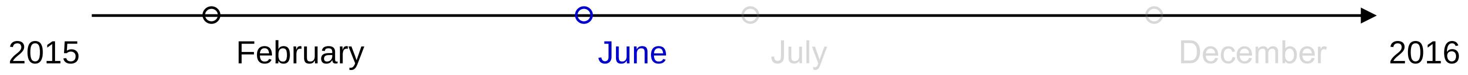
- Outline

1. Background of the CVE-2015-0057 vulnerability
2. A summary of the NCC Group's exploit methods
3. A New CVE-2015-0057 Exploit Technology
4. The Others
5. Reflections and Suggestions

Part Two

Background of the Vulnerability

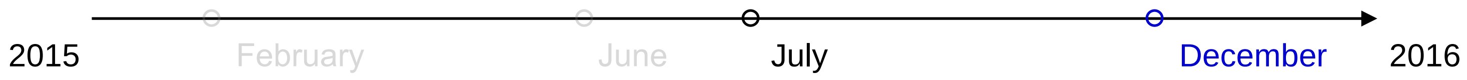
Timeline



- February 10, 2015, Patch Tuesday - Microsoft corporation pushed many patches including CVE-2015-0057. On the same day, Udi Yavo, the CTO of enSilo and the discoverer of the vulnerability, released a technical blog on that topic.

- June 17, 2015, A new variant of the Dyre Banking Trojan was detected by FireEye. This is the first time CVE-2015-0057 was found to be exploited in the wild.

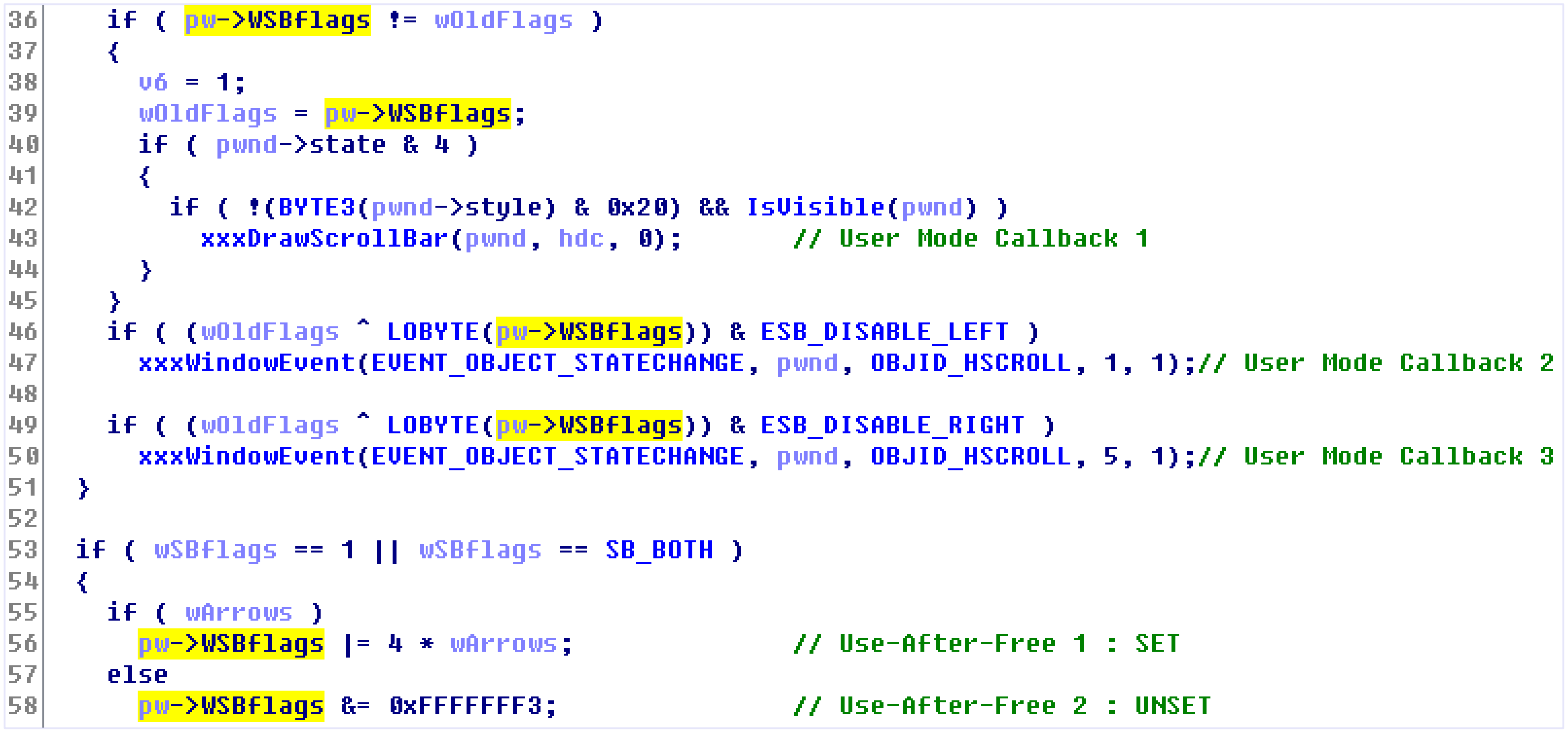
Timeline



- July 8, 2015, NCC Group published their technical blog, describing their 32/64-bit exploit technique in detail.

- December 17, 2015, Jean-Jamil Khalife published his 64-bit exploit code on internet.

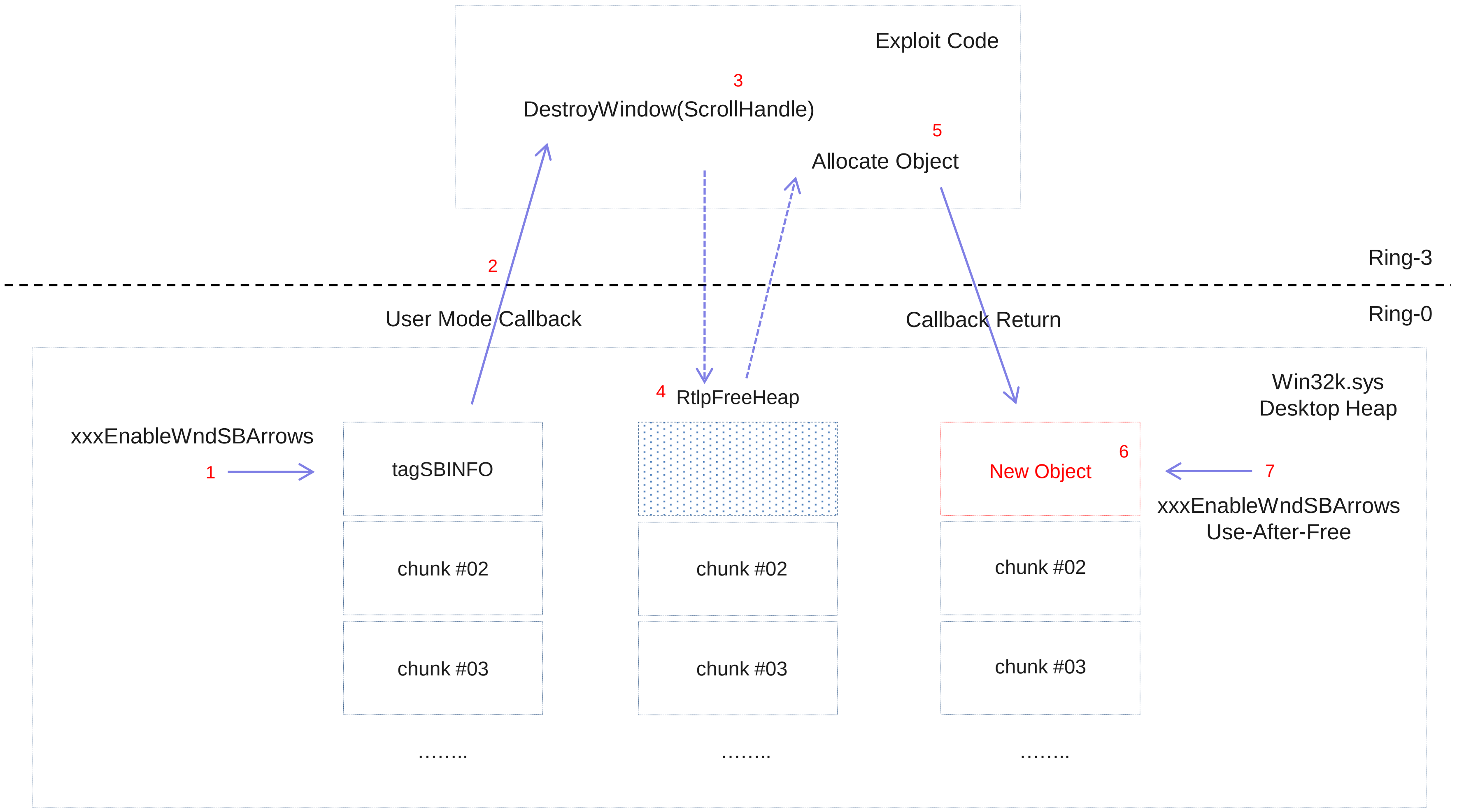
The Vulnerability



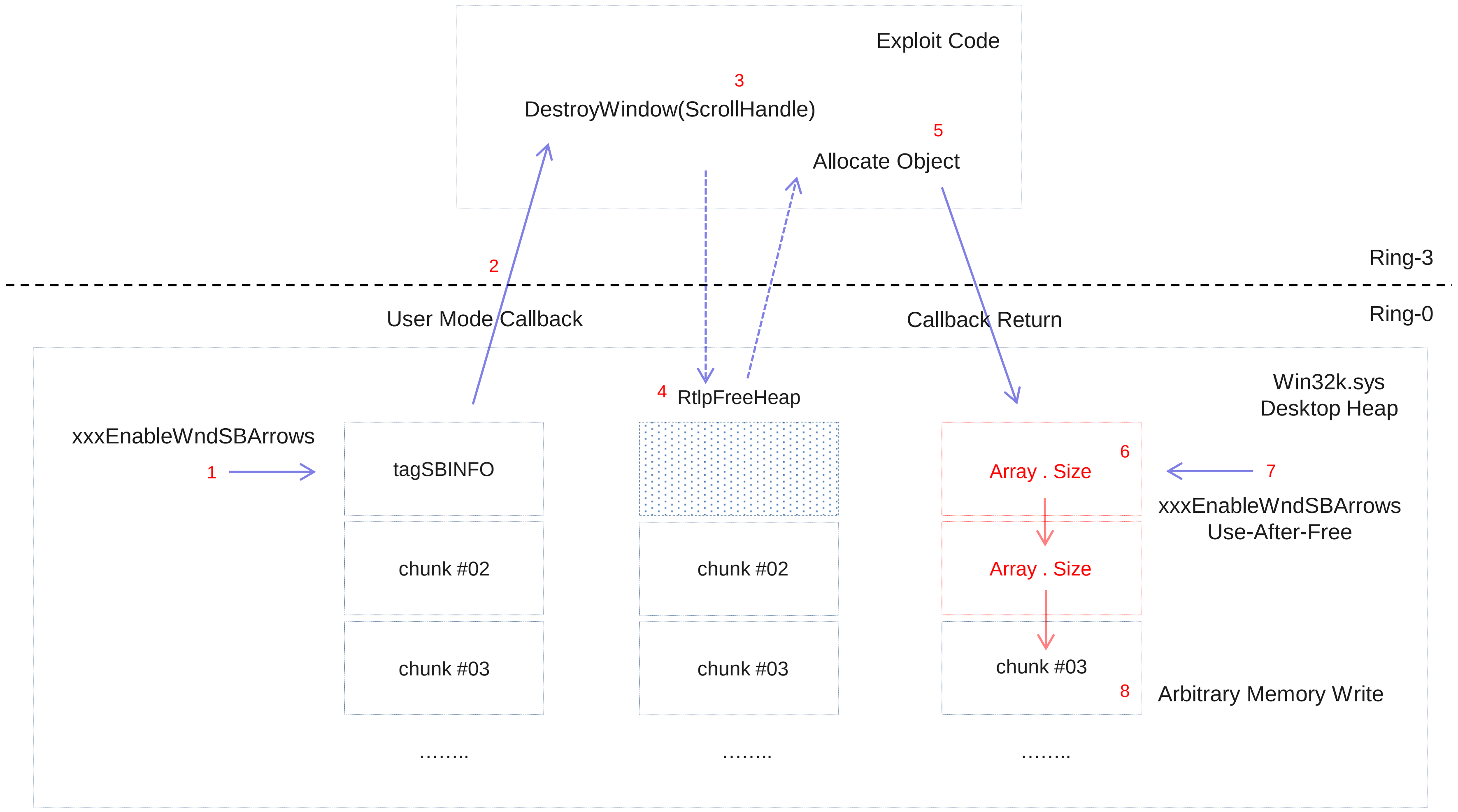
```
    if ( pw->WSBflags != wOldFlags )
    {
      v6 = 1;
      wOldFlags = pw->WSBflags;
      if ( pwnd->state & 4 )
      {
        if ( !(BYTE3(pwnd->style) & 0x20) && IsVisible(pwnd) )
          xxxDrawScrollBar(pwnd, hdc, 0);       // User Mode Callback 1
      }
    }
    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_LEFT )
      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 1, 1);// User Mode Callback 2

    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_RIGHT )
      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 5, 1);// User Mode Callback 3
  }

  if ( wSBflags == 1 || wSBflags == SB_BOTH )
  {
    if ( wArrows )
      pw->WSBflags |= 4 * wArrows;              // Use-After-Free 1 : SET
    else
      pw->WSBflags &= 0xFFFFFFF3;               // Use-After-Free 2 : UNSET
```

Win32k!xxxEnableWndSBArrows

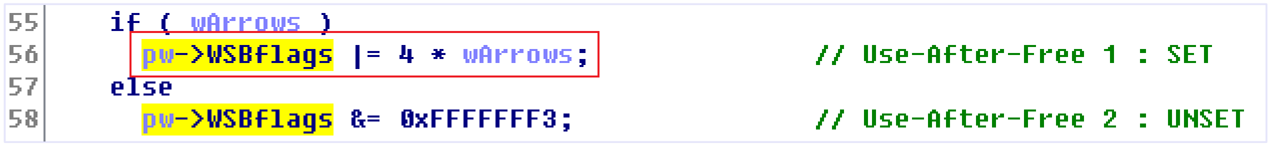
The Old-school Kernel Attack Surface



The Old-school Kernel Attack Surface



Which Object Can Be Used?



```
if ( wArrows )
  pw->WSBflags |= 4 * wArrows;                    // Use-After-Free 1 : SET
else
  pw->WSBflags &= 0xFFFFFFF3;                     // Use-After-Free 2 : UNSET
```

Win32k!xxxEnableWndSBArrows

```
typedef struct tagSBINFO {
    INT WSBflags;
    SBDATA Horz;
    SBDATA Vert;
} SBINFO, *PSBINFO;
```

```
typedef struct tagPROPLIST {
    UINT cEntries;
    UINT iFirstFree;
    tagPROP aprop[1];
} SBINFO, *PSBINFO;
```

tagPROPLIST object

1. tagPROPLIST object and vulnerability related object tagSBINFO, are all allocated from the Win32K Desktop Heap.

2. Inside the tagPROPLIST object, there is an array named tagPROP. This array's size can be adjusted to meet the needs of the exploit.

3. The contents of the tagPROPLIST object can be manipulated through User32 APIs, such as SetProp routine.

4. Crucially, the behavior of SET after the Use-After-Free Vulnerability will overwrite tagPROPLIST’s cEntries field, which means that any subsequent operations on this tagPROPLIST object can result in another out of bounds access.

tagPROP and InternalSetProp Pseudocode

```
typedef struct tagPROP {
    KHANDLE hData;
    ATOM atomKey;
    WORD fs;
} PROP, *PPROP;

BOOL InternalSetProp(LPWSTR pszKey, HANDLE hData, DWORD dwFlags)
{
    PPROP pprop = FindorCreateProp();

    ......

    pprop->atomKey = PTR_TO_ID(pszKey);
    pprop->fs = dwFlags; /* can not be controlled (0/2) */
    pprop->hData = hData;

    ......
}
```

Restrictions

```
0: kd> dd bc65b468 lc
bc65b468 00060006 00080100 00000004 00000004
bc65b478 00bc1040 0000a918 00000000 00002141
bc65b488 00000000 00003141 deadbeef 0002c01a /* 2 bytes are out of control */
```

32-Bit SetProp Restrictions

```
0: kd> dq fffff900`c0841898 l6
fffff900`c0841898 08000003`00010003 00000002`00000002
fffff900`c08418a8 00000000`02f47580 00000000`0000a918
fffff900`c08418b8 deadbeef`deadbeef 00000000`0002c04d /* 6 bytes are out of control */
```

64-Bit SetProp Restrictions

Two Obstacles

1. The write ability of the tagPROPLIST object's SetProp method is restricted, due to the implementation of InternalSetProp.

2. Since the write, and hence repair, capability is limited, continuous memory corruption is unacceptable.

Part Three

NCC Group's 32/64-bit Exploit Method

NCC Group's 32-bit Exploit Method

1. Use the U-A-F tagPROPLIST object's Out-Of-Bounds write capabilities to manipulate Zombie tagPROPLIST's properties.

2. Use the corrupted Zombie tagPROPLIST object to rewrite the adjacent tagWND object's pSBInfo field.

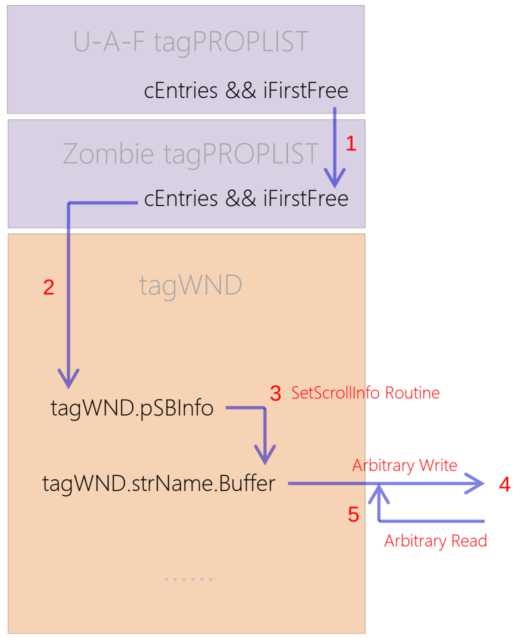
3. Rewrite tagWND object's strName.Buffer field indirectly through the SetScrollInfo routine.

4. tagWND's strName.Buffer field fully under control means that the exploit code achieves arbitrary kernel memory read and write.

Obstacles Solutions

1. Manipulating the tagWND.pSBInfo field by pointing it to the tagWND.strName, and then rewriting tagWND's strName.Buffer field indirectly through SetScrollInfo means obstacle 1 is solved.

2. The full control of the Zombie tagPROPLIST object means obstacle 2 is solved.



NCC Group's 64-bit Exploit Method

1. Overwrite Window Text 1 block's _HEAP_ENTRY based on the U-A-F tagPROPLIST object's Out-Of-Bounds write.

2. Reconstruct an ideal heap layout by using a specially crafted _HEAP_ENTRY for Window Text 1 and another faked _HEAP_ENTRY which is stored in Window Text 2.

3. Due to the fake _HEAP_ENTRY for Window Text 1, a free operation on Window Text 1 causes following tagWND object to be freed as well.

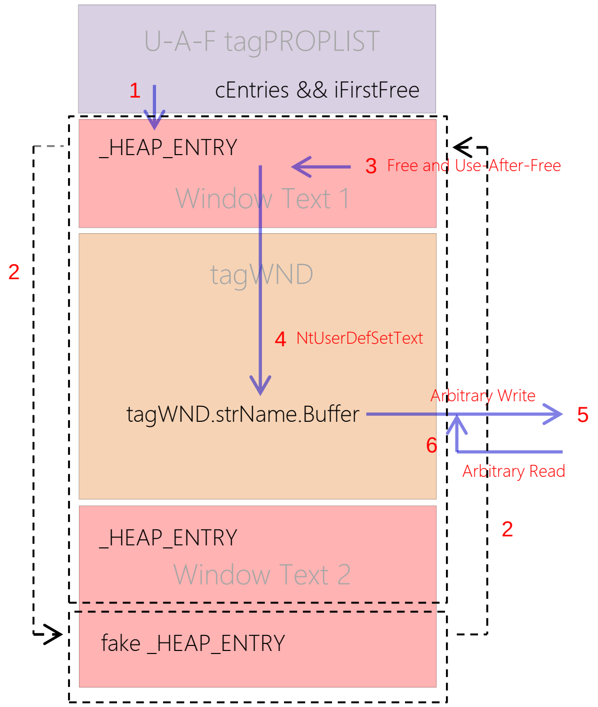
4. Rebuild a new tagWND object based on information leaked from User32!gSharedInfo.

5. Finally, having the tagWND's strName.Buffer field under control means that the exploit code is able to read and write arbitrary kernel memory.

Two Obstacles

1. The faking of heap headers leading to the control of tagWND's strName.Buffer means that obstacle 1 is resolved.

2. The forced Use-After-Free technique and the re-creation of a new tagWND based on kernel information disclosed by User32!gSharedInfo cleverly bypasses obstacle 2.



Part Four

A New CVE-2015-0057 Exploit Technology

32-bit Exploit Method

1. Use the U-A-F tagPROPLIST object's Out-Of-Bounds write capabilities to manipulate Zombie tagPROPLIST's properties.

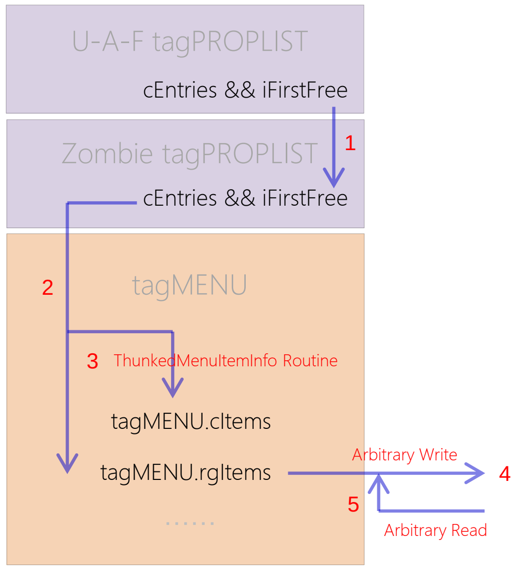
2. Use the corrupted Zombie tagPROPLIST object to rewrite the adjacent tagMENU object's rgItems and cItems fields.

3. tagWND's rgItems field fully under control means that the exploit code achieves arbitrary kernel memory read and write.

Obstacles Solutions

1. The full control of the tagMENU.rgItems and tagMENU.cItems fields means obstacle 1 is solved.

2. The full control of the Zombie tagPROPLIST object means obstacle 2 is solved.



Misaligned tagPROPLIST

```
0: kd> dq fffff900`c085dfc8 l24
fffff900`c085dfc8 08000004`00010003 00000002`0000000e /* U-A-F tagPROPLIST */
fffff900`c085dfd8 00000000`025300b0 00000000`0000a918
fffff900`c085dfe8 00000008`00000008 00000000`00004190

fffff900`c085dff8 08000003`00010003 00000007`00000007 /* Zombie tagPROPLIST */
fffff900`c085e008 00000000`02510600 00000000`0000a918
fffff900`c085e018 00000000`00000000 00000000`00004131

fffff900`c085e028 10000003`00010014 00000000`00020536       /* tagWND */
fffff900`c085e038 00000000`00000003 fffff900`c0723490
fffff900`c085e048 fffffa80`0c3a91d0 fffff900`c085e030
fffff900`c085e058 80000700`40000018 04cf0000`00000100
fffff900`c085e068 00000000`00000000 00000000`02a80000
fffff900`c085e078 fffff900`c085a3d0 fffff900`c08070c0
fffff900`c085e088 fffff900`c0800b90 00000000`00000000
fffff900`c085e098 00000000`00000000 00000000`00000000
fffff900`c085e0a8 00000026`00000084 0000001e`00000008
fffff900`c085e0b8 0000001e`0000007c 00000000`7761946c
fffff900`c085e0c8 fffff900`c083ff70 00000000`00000000
fffff900`c085e0d8 fffff900`c085dfe8 00000000`000000aa  /* tagWND.ppropList */
```

Misaligned tagPROPLIST

```
      0: kd> dq fffff900`c085dfc8 l24
      fffff900`c085dfc8 08000004`00010003 00000002`0000000e /* U-A-F tagPROPLIST */
      fffff900`c085dfd8 00000000`025300b0 00000000`0000a918
+-->  fffff900`c085dfe8 00000008`00000008 00000000`00004190
|
|     fffff900`c085dff8 08000003`00010003 00000007`00000007 /* Zombie tagPROPLIST */
|     fffff900`c085e008 00000000`02510600 00000000`0000a918
|     fffff900`c085e018 00000000`00000000 00000000`00004131
|
|     fffff900`c085e028 10000003`00010014 00000000`00020536       /* tagWND */
|     fffff900`c085e038 00000000`00000003 fffff900`c0723490
|     fffff900`c085e048 fffffa80`0c3a91d0 fffff900`c085e030
|     fffff900`c085e058 80000700`40000018 04cf0000`00000100
|     fffff900`c085e068 00000000`00000000 00000000`02a80000
|     fffff900`c085e078 fffff900`c085a3d0 fffff900`c08070c0
|     fffff900`c085e088 fffff900`c0800b90 00000000`00000000
|     fffff900`c085e098 00000000`00000000 00000000`00000000
|     fffff900`c085e0a8 00000026`00000084 0000001e`00000008
|     fffff900`c085e0b8 0000001e`0000007c 00000000`7761946c
|     fffff900`c085e0c8 fffff900`c083ff70 00000000`00000000
+---  fffff900`c085e0d8 fffff900`c085dfe8 00000000`000000aa  /* tagWND.ppropList */
```

64-bit Exploit Method

1. Use the U-A-F tagPROPLIST object's Out-Of-Bounds write capabilities to manipulate Zombie tagPROPLIST's properties.

2. Use the corrupted Zombie tagPROPLIST object to rewrite the adjacent tagWND object's ppropList field.

3. Point the tagWND.ppropList field to a fake tagPROPLIST object containing use-controlled 'invalid' values.

4. Use the Zombie tagPROPLIST and the misaligned tagPROPLIST object alternately.

5. With tagWND's rgItems field fully under control, the exploit code achieves arbitrary kernel memory read and write.

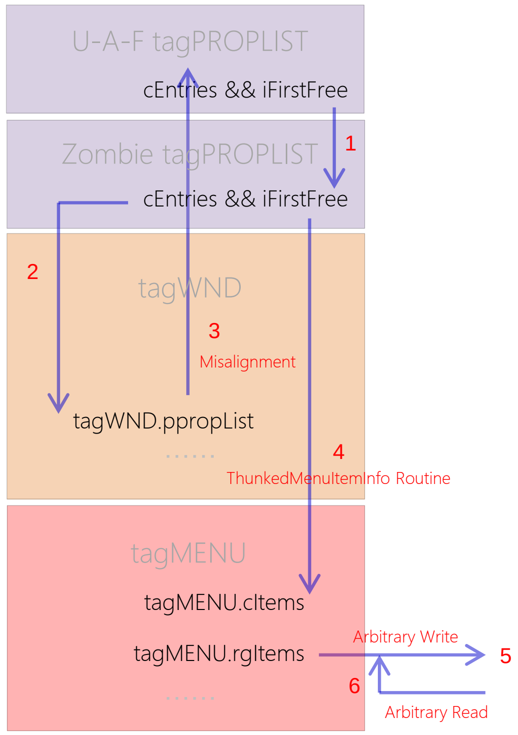
Obstacles Solutions

1. Using the Zombie tagPROPLIST and the crafted, fake tagPROPLIST object alternately to modify the rgItems and cItems fields of tagMENU means that obstacle 1 is solved.

2. Having full control over the cEntries and iFirstFree fields of the Zombie tagPROPLIST object means that obstacle 2 is solved.

64-bit Exploit Method

The write control capability of the Misaligned tagPROPLIST object



DEMO - A New CVE-2015-0057 Exploit Technology

Part Five

The Others

Other Problems

- Heap Data Repair
- Interference from 64-bit Platform's Memory Alignment
- Interference from the CThemeWnd::_AttachInstance

```
0: kd> dq fffff900`c0841f58 fffff900`c0841ff0
fffff900`c0841f58 08000003`00010003 00000002`00000002 /* chunk #01 */
fffff900`c0841f68 00000000`02f7ad30 00000000`0000a918
fffff900`c0841f78 00000000`00000000 00000000`00004136

fffff900`c0841f88 18000003`00010004 00000002`00000002 /* chunk #02 */
fffff900`c0841f98 00000000`02f7b2c0 00000000`0000a918
fffff900`c0841fa8 00000000`00000000 00000000`00004137
fffff900`c0841fb8 00000000`00000000 00000000`00000000 /* alignment */

fffff900`c0841fc8 08000004`00010003 00000002`00000002 /* chunk #03 */
fffff900`c0841fd8 00000000`02f9b850 00000000`0000a918
fffff900`c0841fe8 00000000`00000000 00000000`00004190
```

Dead Code?

"Looking at the code, there are two conditional calls to the function, xxxWindowEvent. These calls are executed only if the old flags of the scrollbar information differ from those of the new flags. However, by the time these conditions appear, the values of the old flags and the new flags are always equal. Hence, the condition for calling xxxWindowEvent is never met. This practically means that this dead-code was there for about 15-years doing absolutely nothing."

http://breakingmalware.com/vulnerabilities/one-bit-rule-bypassing-windows-10-protections-using-single-bit/

Dead Code?

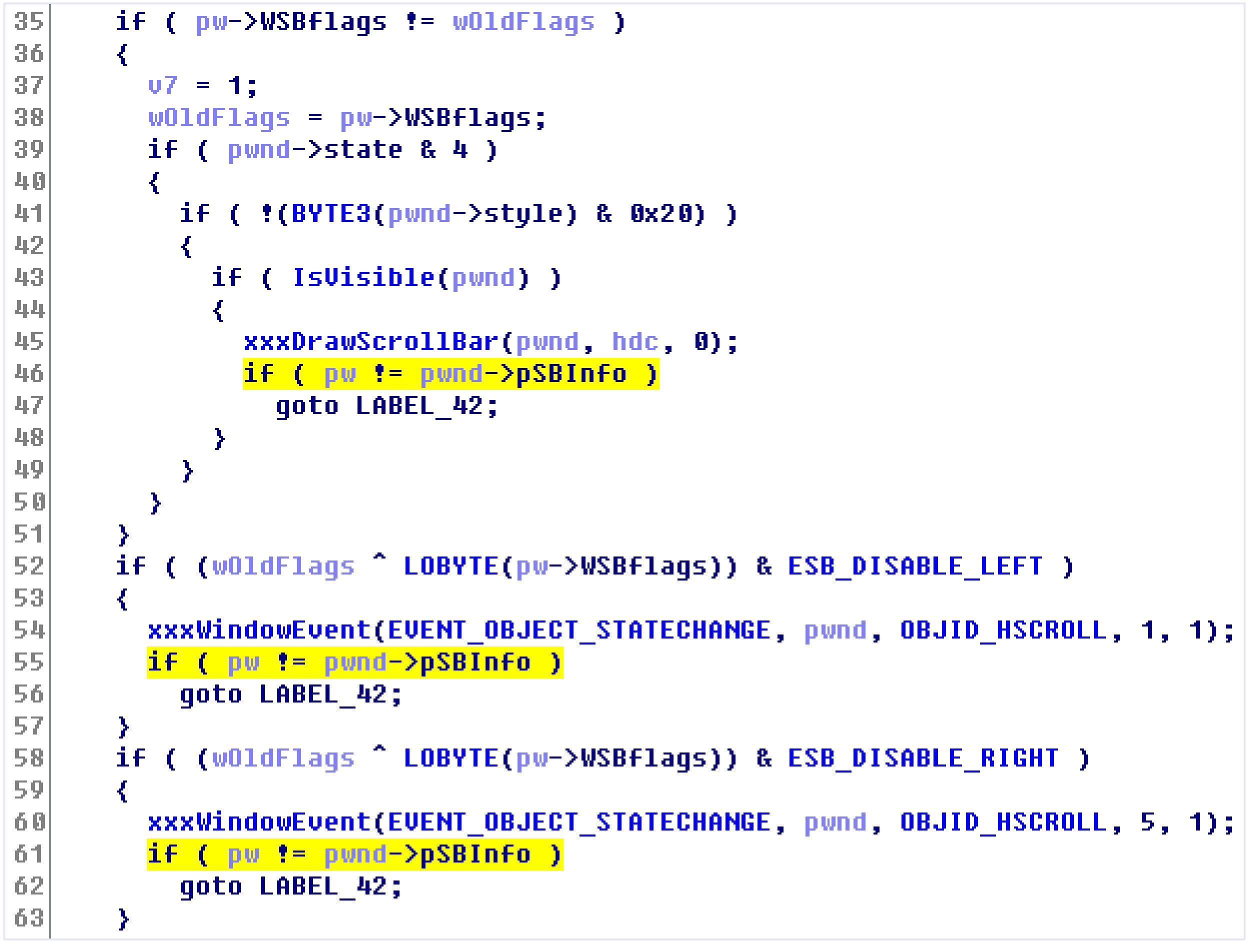
```
/*
 * update the display of the horizontal scroll bar
 */

if (pw->WSBflags != wOldFlags) {
    bRetValue = TRUE;
    wOldFlags = pw->WSBflags;
    if (TestWF(pwnd, WFHPRESENT) && !TestWF(pwnd, WFMINIMIZED) &&
        IsVisible(pwnd)) {
        xxxDrawScrollBar(pwnd, hdc, FALSE);
    }
}

if (FWINABLE()) {
    /* left button */
    if ((wOldFlags & ESB_DISABLE_LEFT) != (pw->WSBflags & ESB_DISABLE_LEFT)) {
        xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
        INDEX_SCROLLBAR_UP, WEF_USEPWNDTHREAD); /* dead-code? */
    }

    /* right button */
    if ((wOldFlags & ESB_DISABLE_RIGHT) != (pw->WSBflags & ESB_DISABLE_RIGHT)) {
         xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
         INDEX_SCROLLBAR_DOWN, WEF_USEPWNDTHREAD); /* dead-code? */
    }
}
```

Win32k!xxxEnableWndSBArrows Pseudocode

Another Trigger Point

```
0: kd> kb
ChildEBP RetAddr  Args to Child
f52838e8 bf8faf21 bc675e20 0012fbcc f5283904 win32k!xxxDestroyWindow
f52838f8 8054160c 00030124 0012fc98 7c92eb94 win32k!NtUserDestroyWindow+0x21
f52838f8 7c92eb94 00030124 0012fc98 7c92eb94 nt!KiFastCallEntry+0xfc
0012fbbc 77d1e672 0042c7ad 00030124 0012fd58 ntdll!KiFastSystemCallRet
0012fc98 77d5906d 00130103 0000800a 00030124 USER32!NtUserDestroyWindow+0xc
0012fcc8 7c92eae3 0012fcd8 00000020 0042a762 USER32! ClientCallWinEventProc+0x2a
0012fcc8 80501a60 0012fcd8 00000020 0042a762 ntdll!KiUserCallbackDispatcher+0x13
f5283bbc 805a1779 f5283c68 f5283c64 bf9a2ccc nt!KiCallUserMode+0x4
f5283c18 bf92c55a 00000050 f5283c44 00000020 nt!KeUserModeCallback+0x87
f5283c88 bf91ccb4 0042a762 e10740a0 bf9a2ccc win32k!xxxClientCallWinEventProc+0x68
f5283cb8 bf8081e8 bf9a2ccc bc675f18 bc675e20 win32k!xxxProcessNotifyWinEvent+0xb9
f5283cfc bf8d136b 0000800a 00000000 fffffffa win32k!xxxWindowEvent+0x182
f5283d2c bf91140e 00000003 00000003 00000003 win32k!xxxEnableWndSBArrows+0xaf
f5283d50 8054160c 00030124 00000003 00000003 win32k!NtUserEnableScrollBar+0x69
f5283d50 7c92eb94 00030124 00000003 00000003 nt!KiFastCallEntry+0xfc
0012fcc8 7c92eae3 0012fcd8 00000020 0042a762 ntdll!KiFastSystemCallRet
0012fcf4 77d6c6ee 5adeb71f 00030124 00000003 ntdll!KiUserCallbackDispatcher+0x13
0012fd14 77d67c01 00030124 00000003 00000003 USER32!NtUserEnableScrollBar+0xc
0012fd54 0042c663 00030124 00000003 00000003 USER32!EnableScrollBar+0x54
0012fe88 0042c807 00400000 80000001 0007ddb4 cve 2015 0057+0x2c663
0012ff60 0042ca4b 00400000 00000000 0015234b cve 2015 0057+0x2c807
0012ffb8 0042c91d 0012fff0 7c816d4f 80000001 cve 2015 0057+0x2ca4b
0012ffc0 7c816d4f 80000001 0007ddb4 7ffdf000 cve 2015 0057+0x2c91d
0012fff0 00000000 0042ad7a 00000000 78746341 kernel32!BaseProcessStart+0x23
```

MS15-010



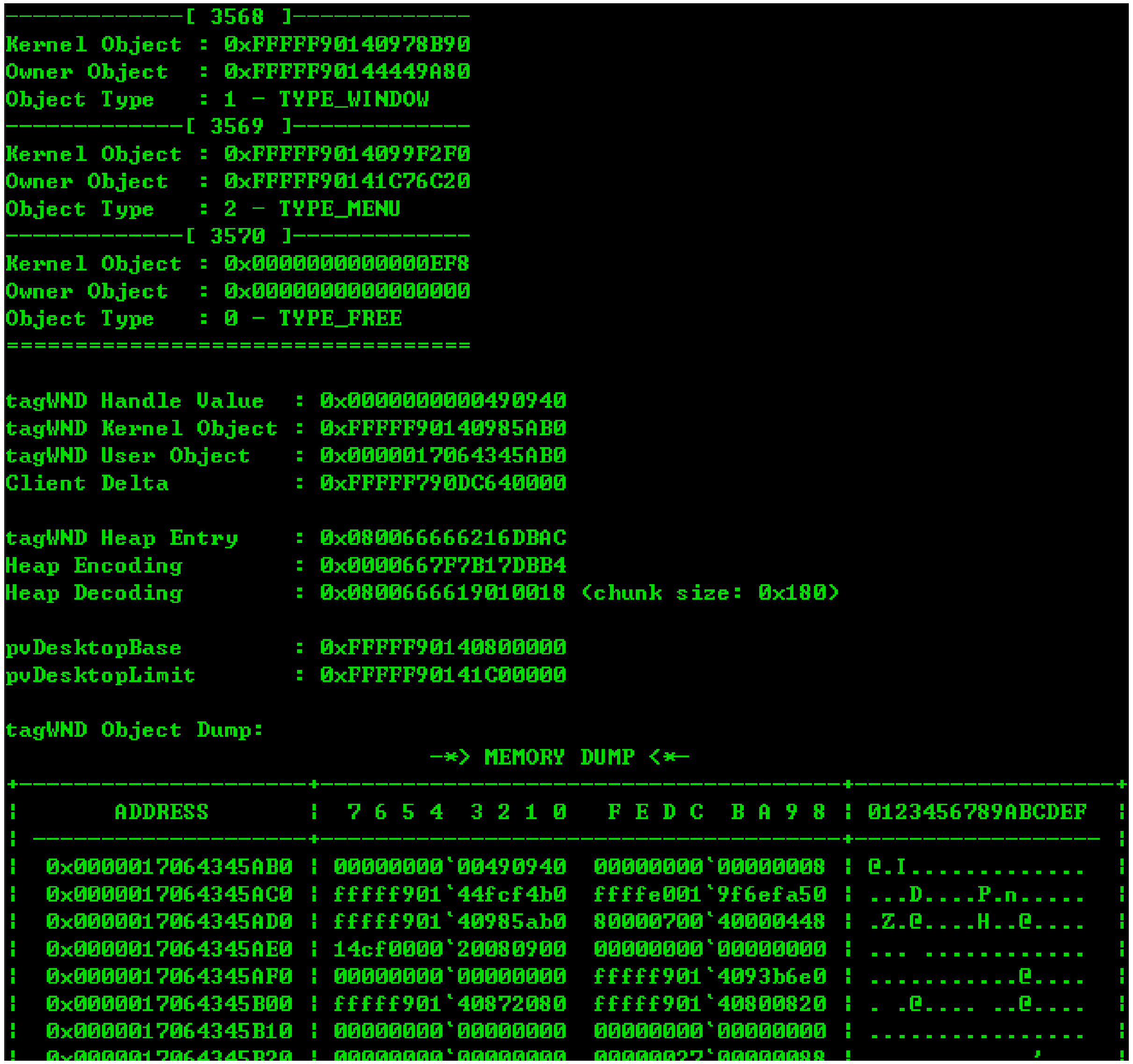
```
if ( pw->WSBflags != wOldFlags )
{
  v7 = 1;
  wOldFlags = pw->WSBflags;
  if ( pwnd->state & 4 )
  {
    if ( !(BYTE3(pwnd->style) & 0x20) )
    {
      if ( IsVisible(pwnd) )
      {
        xxxDrawScrollBar(pwnd, hdc, 0);
        if ( pw != pwnd->pSBInfo )
          goto LABEL_42;
      }
    }
  }
}
if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_LEFT )
{
  xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 1, 1);
  if ( pw != pwnd->pSBInfo )
    goto LABEL_42;
}
if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_RIGHT )
{
  xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 5, 1);
  if ( pw != pwnd->pSBInfo )
    goto LABEL_42;
}
```

Win32k!xxxEnableWndSBArrows, MS15-010

User32!gSharedInfo

- Kernel Routine Address Information
- Kernel Data Structure Information
- Kernel Object Memory Layout Information
- Desktop Heap Information
- Kernel _HEAP_ENTRY Random Cookie Information

User32!gSharedInfo



```
-------------[ 3568 ]-------------
Kernel Object : 0xFFFFF90140978B90
Owner Object  : 0xFFFFF90144449A80
Object Type   : 1 - TYPE_WINDOW
-------------[ 3569 ]-------------
Kernel Object : 0xFFFFF9014099F2F0
Owner Object  : 0xFFFFF90141C76C20
Object Type   : 2 - TYPE_MENU
-------------[ 3570 ]-------------
Kernel Object : 0x0000000000000EF8
Owner Object  : 0x0000000000000000
Object Type   : 0 - TYPE_FREE
==================================

tagWND Handle Value   : 0x0000000000490940
tagWND Kernel Object  : 0xFFFFF90140985AB0
tagWND User Object    : 0x0000017064345AB0
Client Delta          : 0xFFFFF790DC640000

tagWND Heap Entry     : 0x080066666216DBAC
Heap Encoding         : 0x0000667F7B17DBB4
Heap Decoding         : 0x0800666619010018 (chunk size: 0x180)

pvDesktopBase         : 0xFFFFF90140800000
pvDesktopLimit        : 0xFFFFF90141C00000

tagWND Object Dump:
                               -*> MEMORY DUMP <*-
+---------------------+---------------------------------------+--------------------+
¦       ADDRESS       ¦  7 6 5 4  3 2 1 0   F E D C  B A 9 8 ¦ 0123456789ABCDEF   ¦
¦ --------------------+---------------------------------------+------------------  ¦
¦  0x0000017064345AB0 ¦ 00000000`00490940  00000000`00000008 ¦ @.I.............   ¦
¦  0x0000017064345AC0 ¦ fffff901`44fcf4b0  ffffe001`9f6efa50 ¦ ...D....P.n.....   ¦
¦  0x0000017064345AD0 ¦ fffff901`40985ab0  80000700`40000448 ¦ .Z.@....H..@....   ¦
¦  0x0000017064345AE0 ¦ 14cf0000`20080900  00000000`00000000 ¦ ... ............   ¦
¦  0x0000017064345AF0 ¦ 00000000`00000000  fffff901`4093b6e0 ¦ ...........@....   ¦
¦  0x0000017064345B00 ¦ fffff901`40872080  fffff901`40800820 ¦ . .@.... ..@....   ¦
¦  0x0000017064345B10 ¦ 00000000`00000000  00000000`00000000 ¦ ................   ¦
¦  0x0000017064345B20 ¦ 00000000`00000000  00000027`00000088 ¦            '       ¦
```

Part Six

The End

Think Deeply

- Win32K subsystem's lock mechanism
- The life cycle management of window related objects
- User32!gSharedInfo sharing mechanism
- State-Inconsistency type vulnerability

Acknowledgements

Yin Hong Chang	Joonho Sa
FireEye Labs ZDC	FireEye Labs SG

Q&A

yu.wang@fireeye.com