How to find the vulnerability to bypass the Control Flow Guard

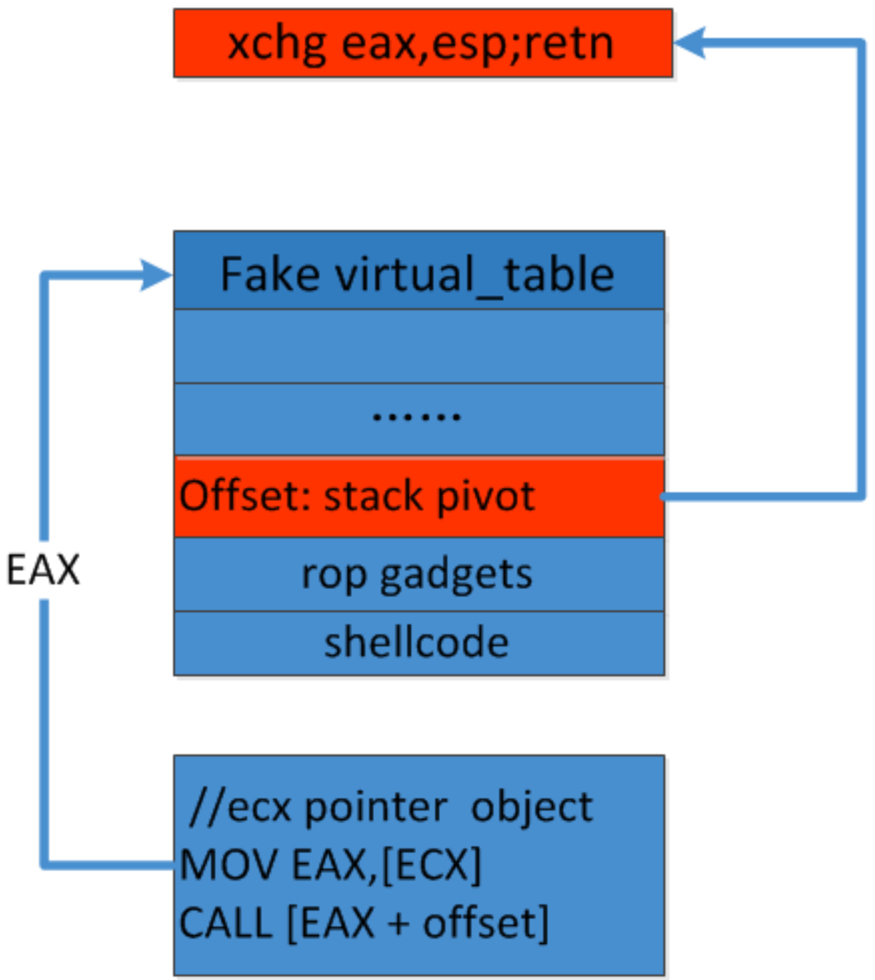
Henry Li(@zenhumany)

- Trend Micro CDC Zeroday discovery Team
- Security Researcher
- Six Years Experience
- Expert in browser 0day vulnerability analysis, discovery and exploit.
- Won the Microsoft Mitigation Bypass Bounty in 2016
- Won the Microsoft Edge Web Platform on WIP Bounty
- MSRC Top 17 in year 2016
- twitter/weibo: zenhumany

TREND MICRO™

Why we need CFG bypass vulnerability

Before CFG



xchg eax,esp;retn

Fake virtual_table

……

Offset: stack pivot

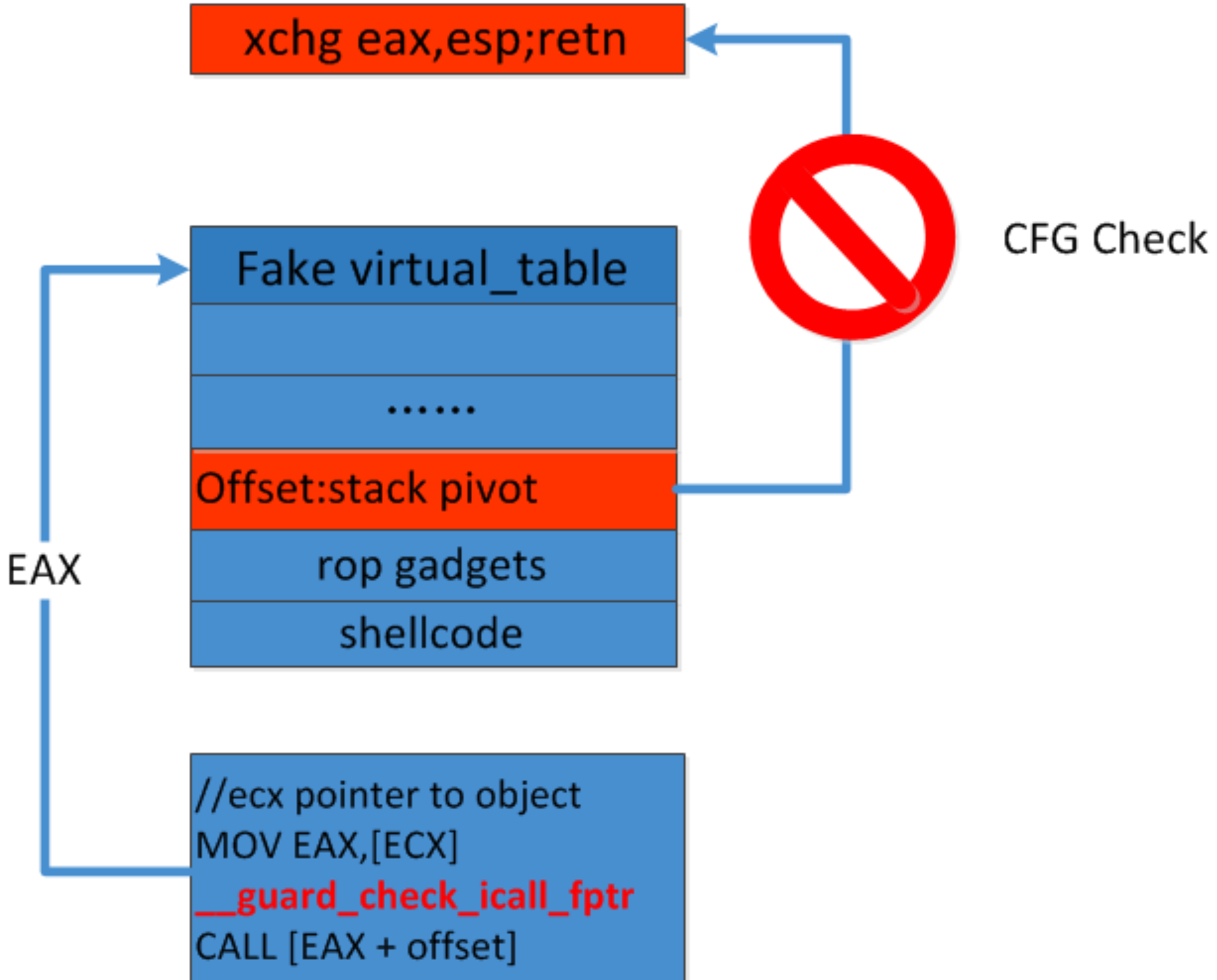
rop gadgets

shellcode

EAX

```
//ecx pointer  object
MOV EAX,[ECX]
CALL [EAX + offset]
```

After CFG



xchg eax,esp;retn

CFG Check

Fake virtual_table

……

Offset:stack pivot

rop gadgets

shellcode

EAX

```
//ecx pointer to object
MOV EAX,[ECX]
__guard_check_icall_fptr
CALL [EAX + offset]
```

Why we need CFG bypass vulnerability

- Even your have arbitrary read/write vulnerability, you need bypass CFG to run shellcode
- No universal CFG bypass method

Agenda

- **Attack Surface**
- **Find vulnerability**
- **Exploit Framework**
- **Improvements**

Attack Surface

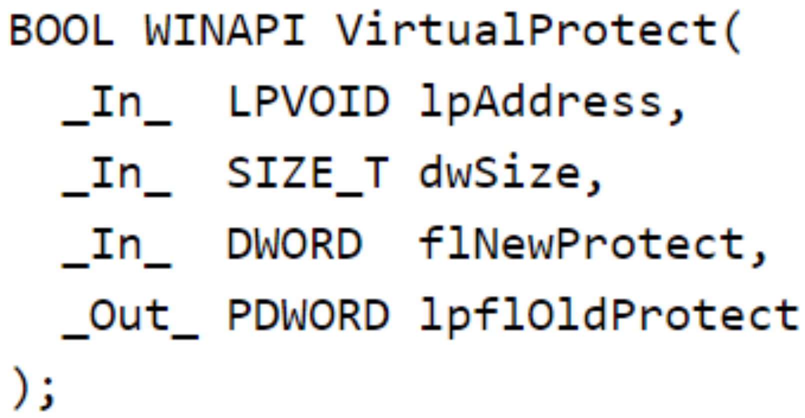
- CFG attribute Change Functions
- write return address
- No Control Flow Guard check
- CFG sensitive API

Attack Surface 1

- **CFG ATTRIBUTE CHANGE FUNCTIONS**
  - **VirtualAlloc**
  - **VirtualProtect**
  - **SetProcessValidCallTargets**

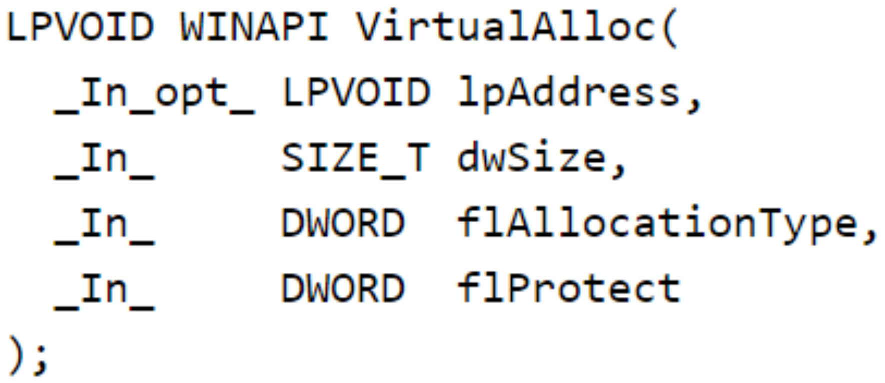
VirtualProtect-VirtualAlloc

- **VirtualProtect**
  - flNewProtect 0x40
    - Memory Protection PAGE_EXECUTE_READWRITE
    - The address in the pages are all CFG valid
  - flNewProtect 0x40000040
    - Memory Protection PAGE_EXECUTE_READWRITE
    - The address in the pages are all CFG invalid



```
BOOL WINAPI VirtualProtect(
  _In_  LPVOID lpAddress,
  _In_  SIZE_T dwSize,
  _In_  DWORD  flNewProtect,
  _Out_ PDWORD lpflOldProtect
);
```

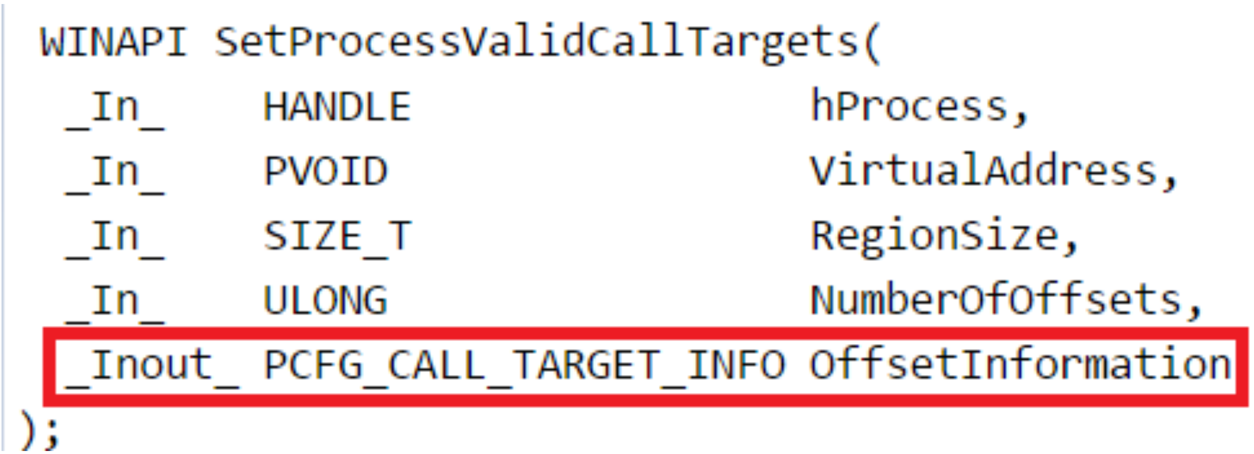
- **VirtualAlloc**
  - flProtect 0x40
    - Memory Protection PAGE_EXECUTE_READWRITE
    - The address in the pages are all CFG valid
  - flProtect 0x40000040
    - Memory Protection PAGE_EXECUTE_READWRITE
    - The address in the pages are all CFG invalid



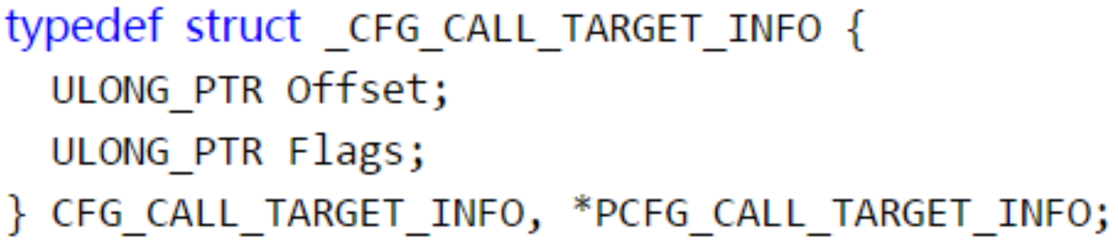
```
LPVOID WINAPI VirtualAlloc(
  _In_opt_ LPVOID lpAddress,
  _In_     SIZE_T dwSize,
  _In_     DWORD  flAllocationType,
  _In_     DWORD  flProtect
);
```

SetProcessValidCallTargets

- **SetProcessValidCallTargets**
  - Flags
    - **CFG_CALL_TARGET_VALID**
    - Otherwise, it will be marked as invalid

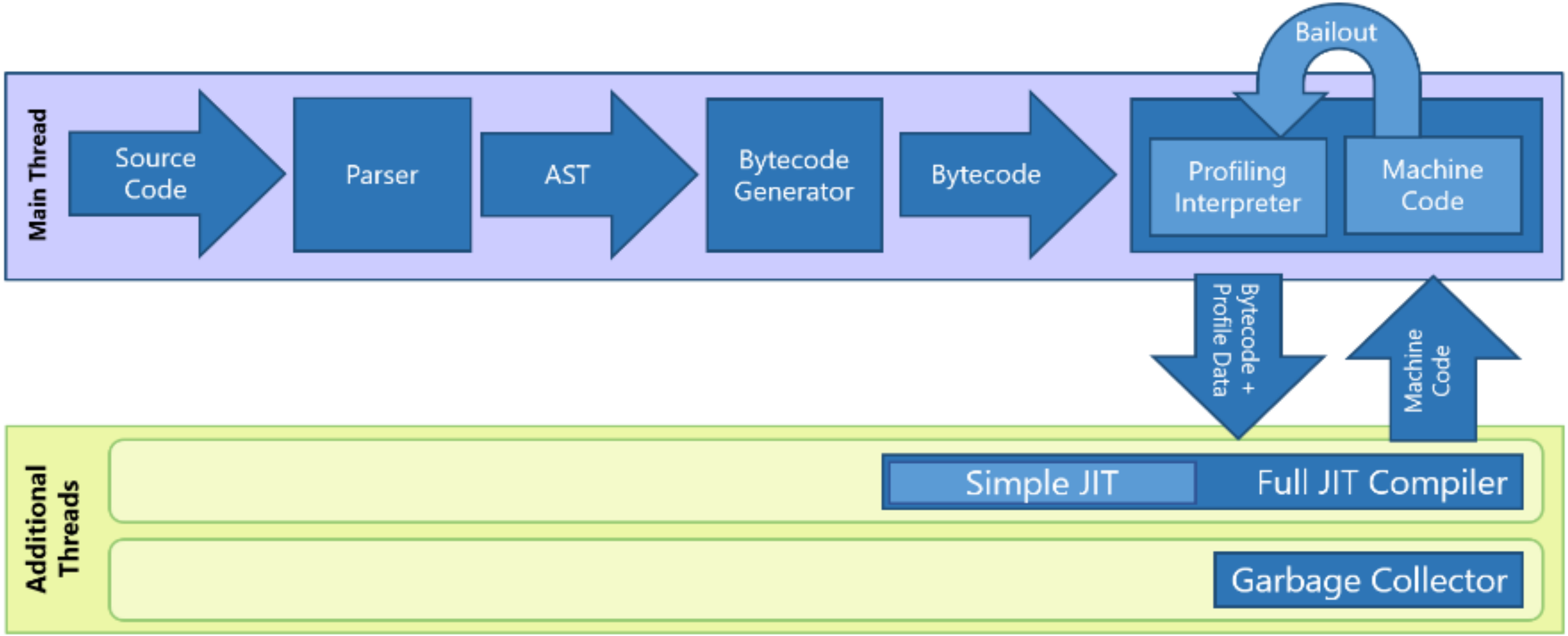


```
WINAPI SetProcessValidCallTargets(
  _In_     HANDLE                 hProcess,
  _In_     PVOID                  VirtualAddress,
  _In_     SIZE_T                 RegionSize,
  _In_     ULONG                  NumberOfOffsets,
  _Inout_  PCFG_CALL_TARGET_INFO  OffsetInformation
);
```

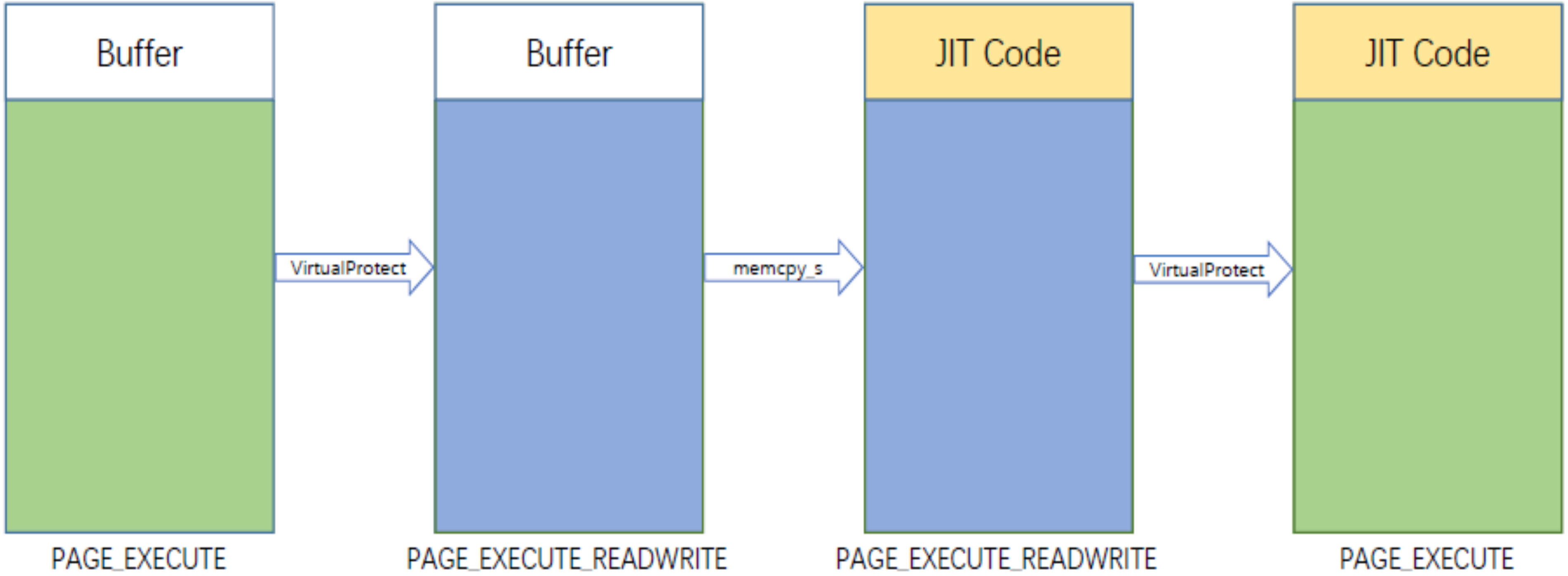


```
typedef struct _CFG_CALL_TARGET_INFO {
  ULONG_PTR Offset;
  ULONG_PTR Flags;
} CFG_CALL_TARGET_INFO, *PCFG_CALL_TARGET_INFO;
```

Chakra Engine Architecture



JIT Memory Management



Attack Surface 1

- **In Microsoft Edge, there are two types of JIT:**
  - javascript JIT, in the **chakra.dll Module**.
  - SHADER JIT, in the **d3d10warp.dll Module**.

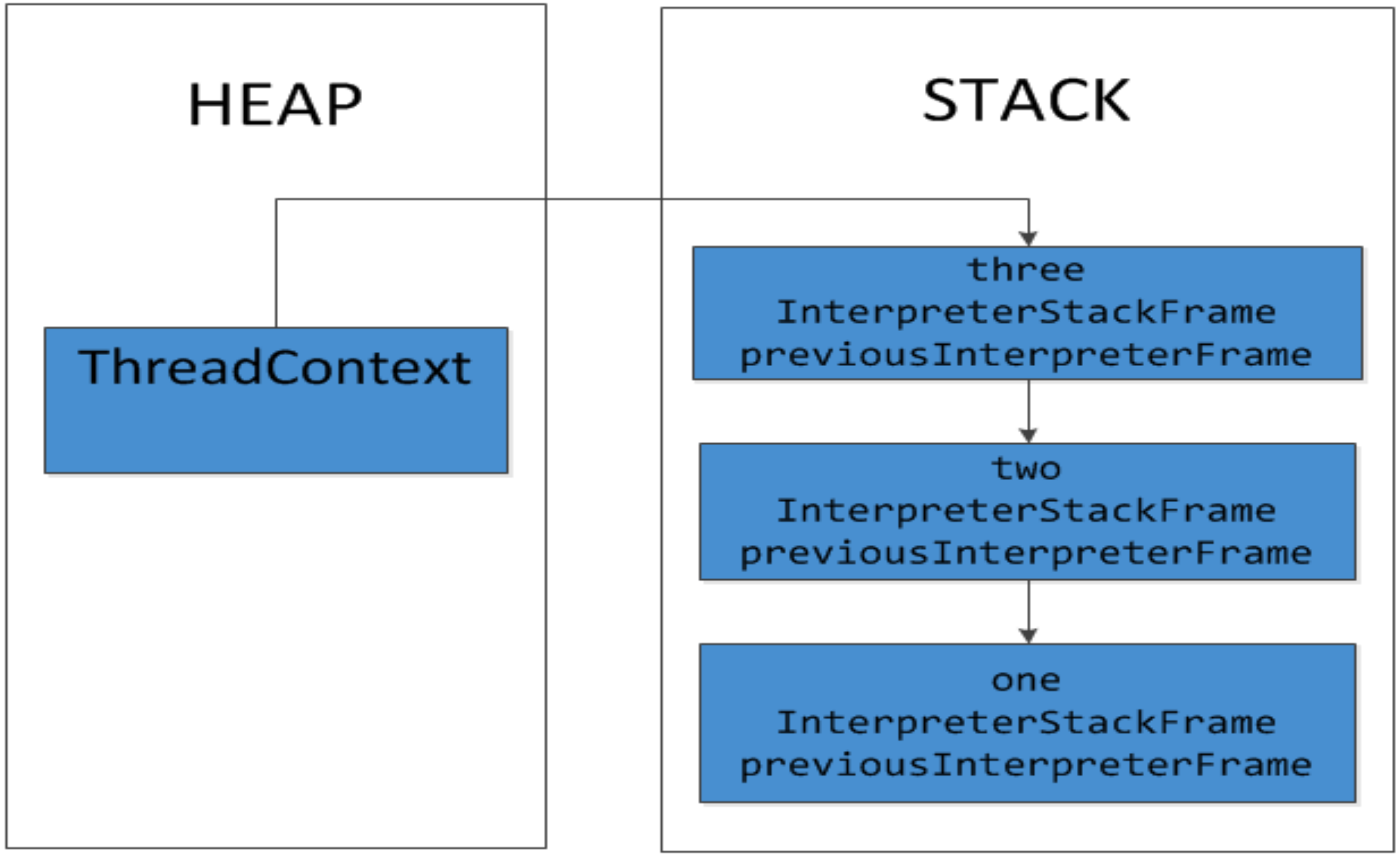
Attack Surface 2
write the return address

- Because the CFG does not check the ret, we can write the return address to bypass the CFG.

- In chakra engine, the interpreting execution mode will simulate a function call stack. The implementation will save some stackframe information on a special object in the heap.

- If we have arbitrary read and write vulnerability, we may can infoleak some stack information.

Interpreter StackFrame



```
function one( )
{
   two( )
}

function two( )
{
   three( )
}

function three( )
{
    return 0x8000;
}
```

Attack Surface 3
Indirect call with no CFG check

- **JIT code is implemented in the runtime.**
- **The CFG support in JIT may be manual maintenance.**
- **Pay attention to the JIT code to find indirect call with no CFG check.**

Attack Surface 4
CFG Sensitive API

- Use these function to bypass CFG
  - VirtualProtect
  - VirtualAlloc
  - longjmp/setjmp
  - ……

Find Vulnerability

- **Six CFG bypass vulnerabilities**

Notes:

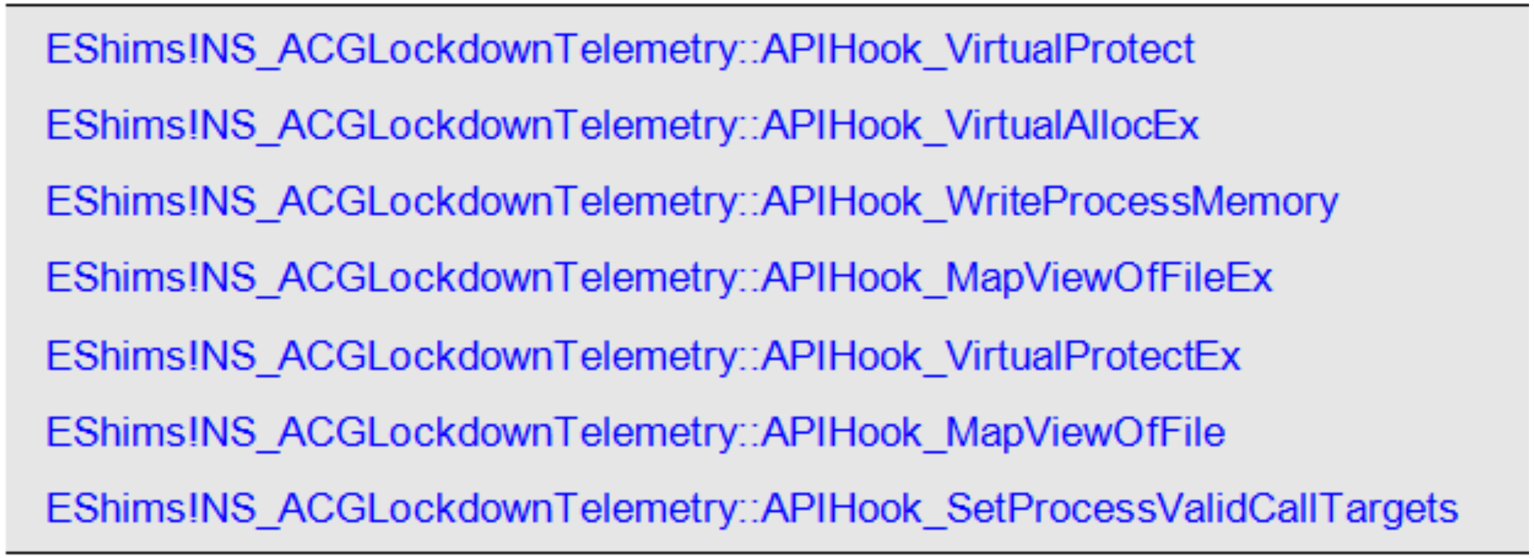
All of the following bypass vulnerabilities suppose you have arbitrary read/write vulnerability

Vuln 1

- **eshims!VirtualProtect to bypass CFG and DEP**
- **Vuln Type: Call Sensitive API out of context**
- **Module: Eshims**
- **Operation System: Windows 10 14367 32 bit**
- **BYPASS CFG/DEP**

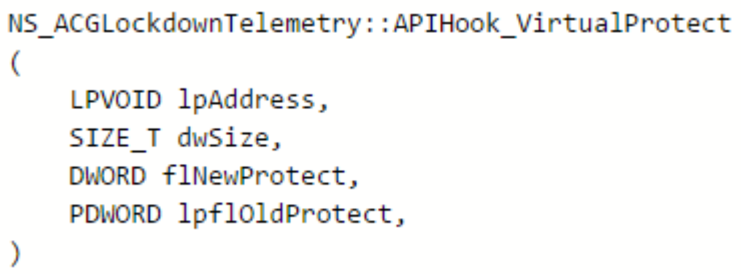
Vuln 1

- eshims.dll is a module in Microsoft Edge
- eshims have following hook functios,the functions are CFG valid.

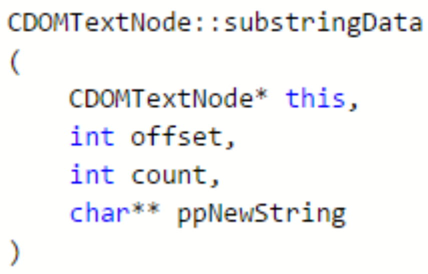


```
EShims!NS_ACGLockdownTelemetry::APIHook_VirtualProtect
EShims!NS_ACGLockdownTelemetry::APIHook_VirtualAllocEx
EShims!NS_ACGLockdownTelemetry::APIHook_WriteProcessMemory
EShims!NS_ACGLockdownTelemetry::APIHook_MapViewOfFileEx
EShims!NS_ACGLockdownTelemetry::APIHook_VirtualProtectEx
EShims!NS_ACGLockdownTelemetry::APIHook_MapViewOfFile
EShims!NS_ACGLockdownTelemetry::APIHook_SetProcessValidCallTargets
```

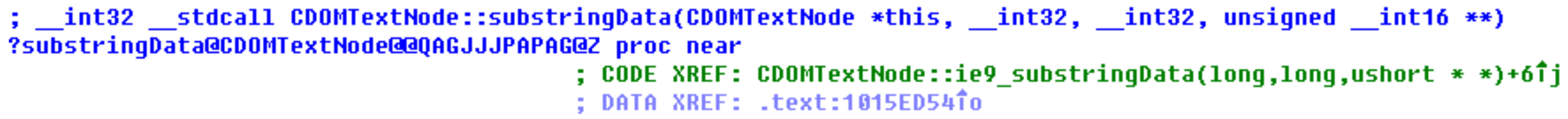
Vuln 1



```
NS_ACGLockdownTelemetry::APIHook_VirtualProtect
(
    LPVOID lpAddress,
    SIZE_T dwSize,
    DWORD flNewProtect,
    PDWORD lpflOldProtect,
)
```

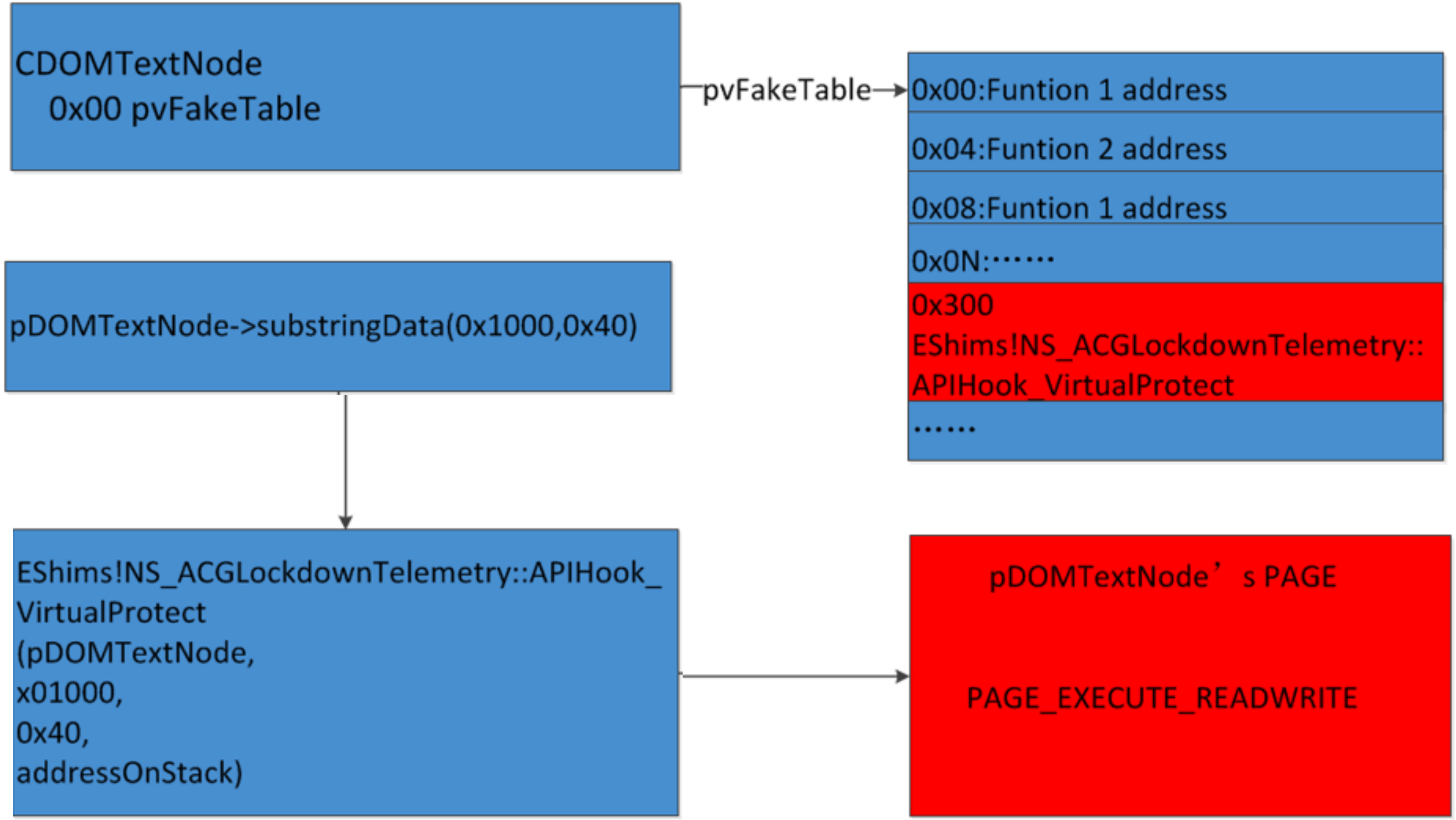


```
CDOMTextNode::substringData
(
    CDOMTextNode* this,
    int offset,
    int count,
    char** ppNewString
)
```



```
; __int32 __stdcall CDOMTextNode::substringData(CDOMTextNode *this, __int32, __int32, unsigned __int16 **)
?substringData@CDOMTextNode@@QAGJJJPAPAG@Z proc near
                                        ; CODE XREF: CDOMTextNode::ie9_substringData(long,long,ushort * *)+6↑j
                                        ; DATA XREF: .text:1015ED54↑o
```

Vuln 1: Exploit Method



CDOMTextNode
0x00 pvFakeTable

pvFakeTable

0x00:Funtion 1 address

0x04:Funtion 2 address

0x08:Funtion 1 address

0x0N:……

0x300
EShims!NS_ACGLockdownTelemetry::
APIHook_VirtualProtect

……

pDOMTextNode->substringData(0x1000,0x40)

EShims!NS_ACGLockdownTelemetry::APIHook_
VirtualProtect
(pDOMTextNode,
x01000,
0x40,
addressOnStack)

pDOMTextNode’ s PAGE

PAGE_EXECUTE_READWRITE

Vuln 2

- CodeStorageBlock::Protect function to bypass CFG and DEP
- Vuln Type:Call Sensitive API out of context
- Module: D3D10Warp.dll
- Operation System: Windows 10 14393.5 32 bit
- BYPASS CFG/DEP

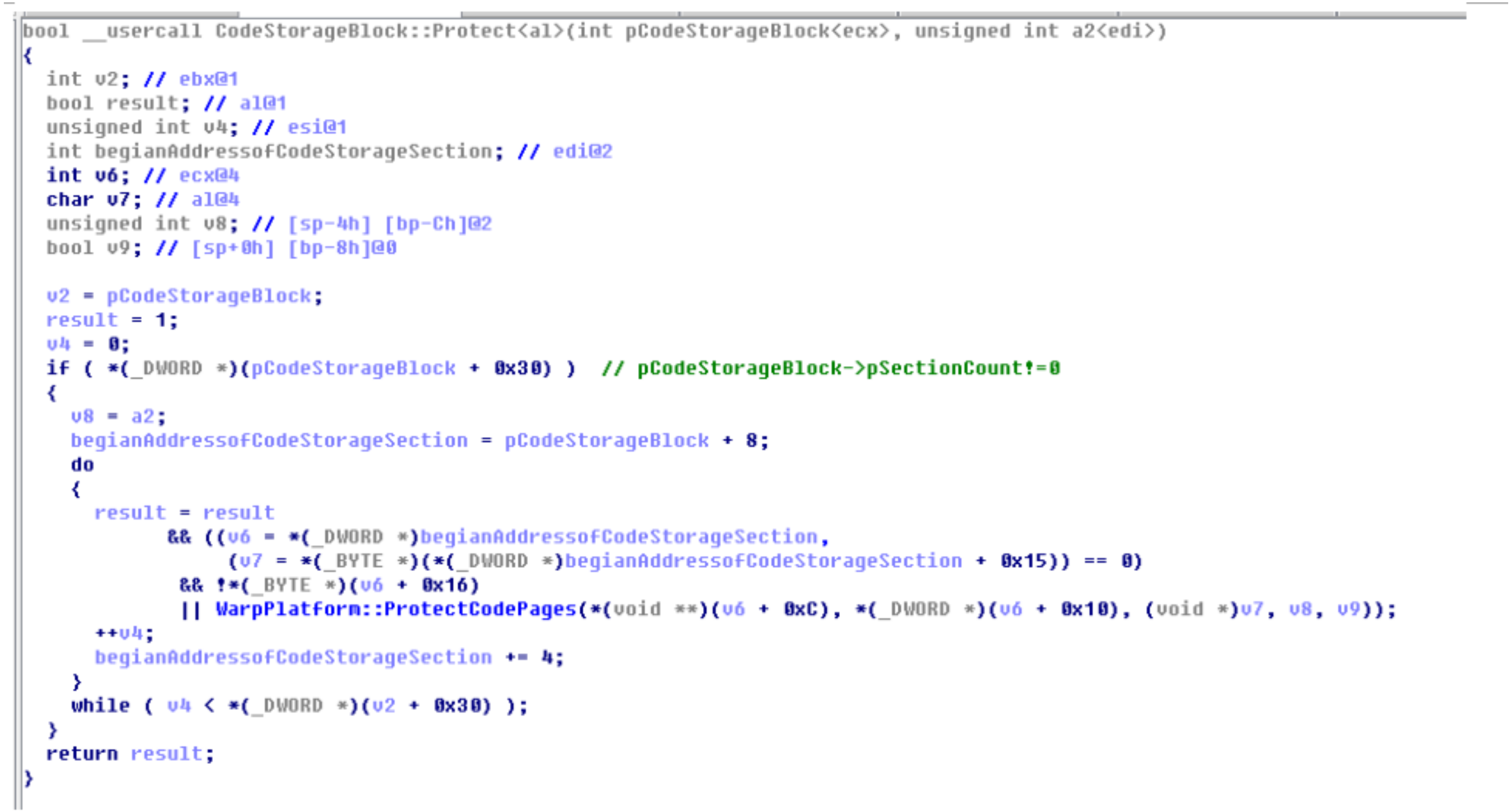
Vuln 2

- CodeStorageBlock::Protect is CFG valid

```
CodeStorageBlock(0x38)
        0x00 pVtable
        0x04 pCodeStorage
    0x08 begianAddressofCodeStorageSection
        0x30 pSectionCount
```

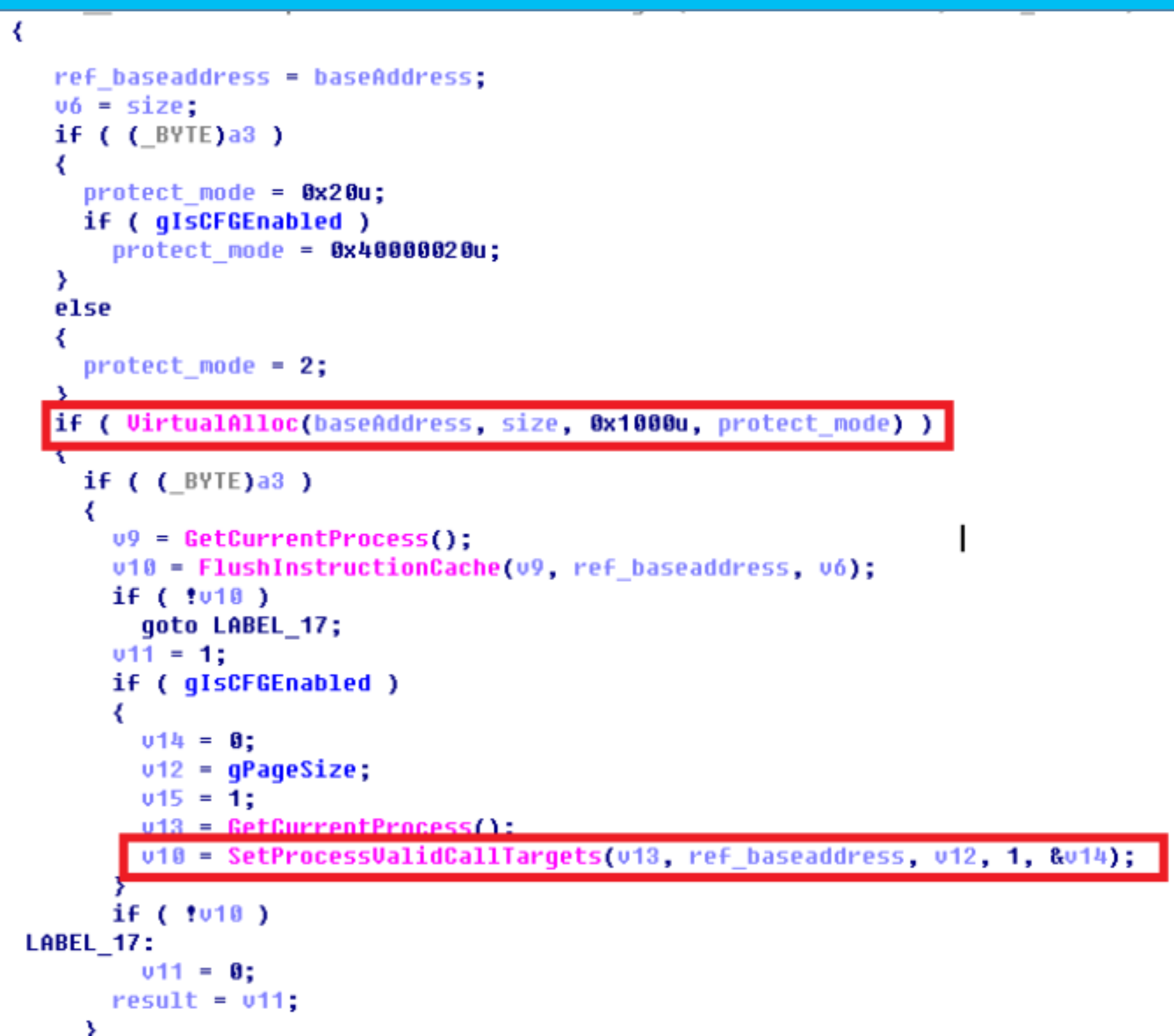
```
CodeStorageSection(0x18)
        0x00 pCodeStorageChunk
        0x04 pPrevCodeStorageSection
        0x08 pNextCodeStorageSection
        0x0c baseAddress
        0x10 size
        0x14 flag_busy :byte
```

Vuln 2



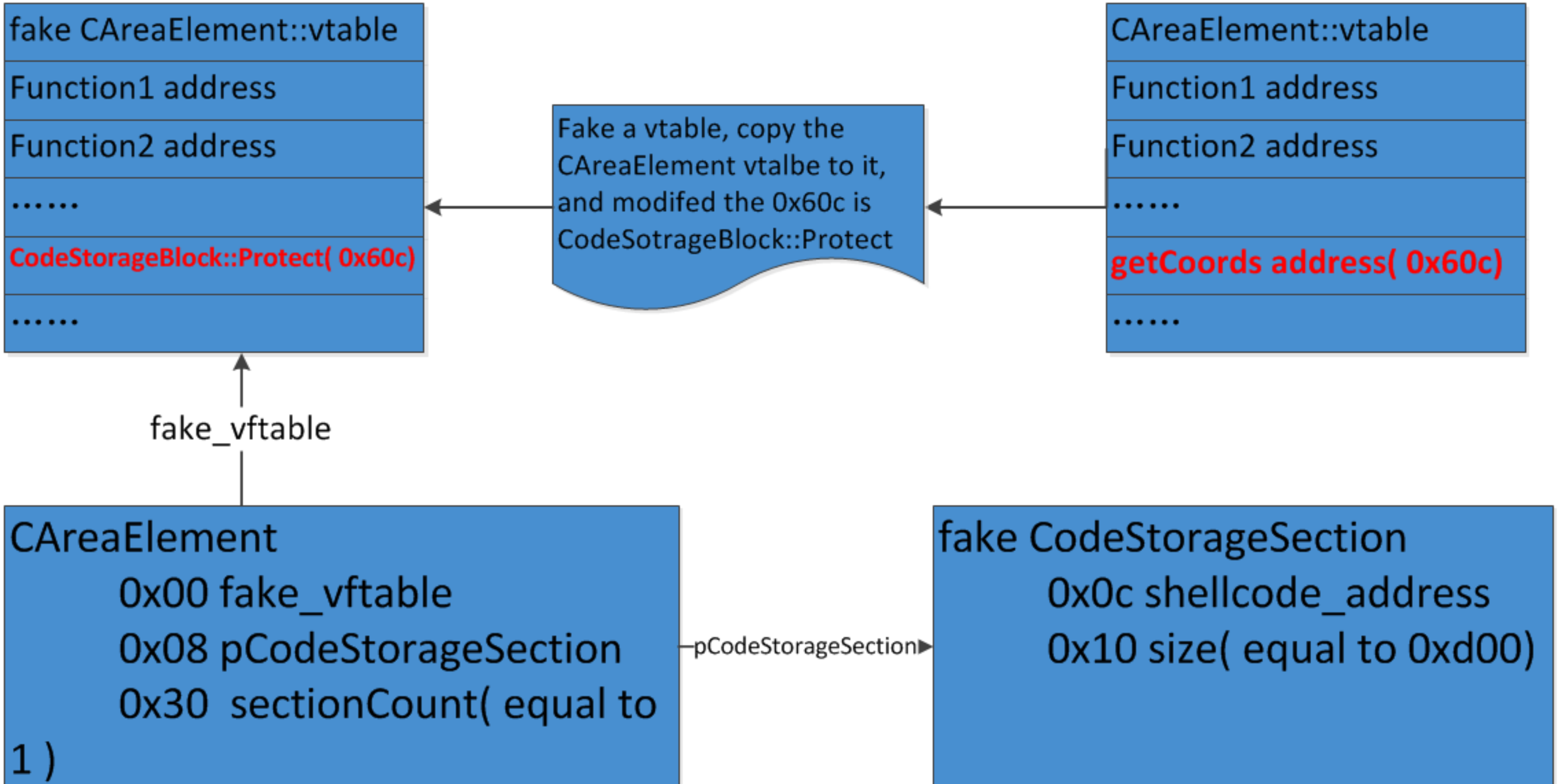
```
bool __usercall CodeStorageBlock::Protect<al>(int pCodeStorageBlock<ecx>, unsigned int a2<edi>)
{
  int v2; // ebx@1
  bool result; // al@1
  unsigned int v4; // esi@1
  int begianAddressofCodeStorageSection; // edi@2
  int v6; // ecx@4
  char v7; // al@4
  unsigned int v8; // [sp-4h] [bp-Ch]@2
  bool v9; // [sp+0h] [bp-8h]@0

  v2 = pCodeStorageBlock;
  result = 1;
  v4 = 0;
  if ( *(_DWORD *)(pCodeStorageBlock + 0x30) )  // pCodeStorageBlock->pSectionCount!=0
  {
    v8 = a2;
    begianAddressofCodeStorageSection = pCodeStorageBlock + 8;
    do
    {
      result = result
            && ((v6 = *(_DWORD *)begianAddressofCodeStorageSection,
                 (v7 = *(_BYTE *)(*(_DWORD *)begianAddressofCodeStorageSection + 0x15)) == 0)
             && !*(_BYTE *)(v6 + 0x16)
             || WarpPlatform::ProtectCodePages(*(void **)(v6 + 0xC), *(_DWORD *)(v6 + 0x10), (void *)v7, v8, v9));
      ++v4;
      begianAddressofCodeStorageSection += 4;
    }
    while ( v4 < *(_DWORD *)(v2 + 0x30) );
  }
  return result;
}
```

Vuln 2



```
{
  ref_baseaddress = baseAddress;
  v6 = size;
  if ( (_BYTE)a3 )
  {
    protect_mode = 0x20u;
    if ( gIsCFGEnabled )
      protect_mode = 0x40000020u;
  }
  else
  {
    protect_mode = 2;
  }
  if ( VirtualAlloc(baseAddress, size, 0x1000u, protect_mode) )
  {
    if ( (_BYTE)a3 )
    {
      v9 = GetCurrentProcess();
      v10 = FlushInstructionCache(v9, ref_baseaddress, v6);
      if ( !v10 )
        goto LABEL_17;
      v11 = 1;
      if ( gIsCFGEnabled )
      {
        v14 = 0;
        v12 = gPageSize;
        v15 = 1;
        v13 = GetCurrentProcess();
        v10 = SetProcessValidCallTargets(v13, ref_baseaddress, v12, 1, &v14);
      }
      if ( !v10 )
 LABEL_17:
        v11 = 0;
      result = v11;
    }
```

Vuln 2:Exploit Method



fake CAreaElement::vtable

Function1 address

Function2 address

……

CodeStorageBlock::Protect(0x60c)

……

Fake a vtable, copy the CAreaElement vtalbe to it, and modifed the 0x60c is CodeSotrageBlock::Protect

CAreaElement::vtable

Function1 address

Function2 address

……

getCoords address(0x60c)

……

fake_vftable

CAreaElement
0x00 fake_vftable
0x08 pCodeStorageSection
0x30 sectionCount(equal to 1)

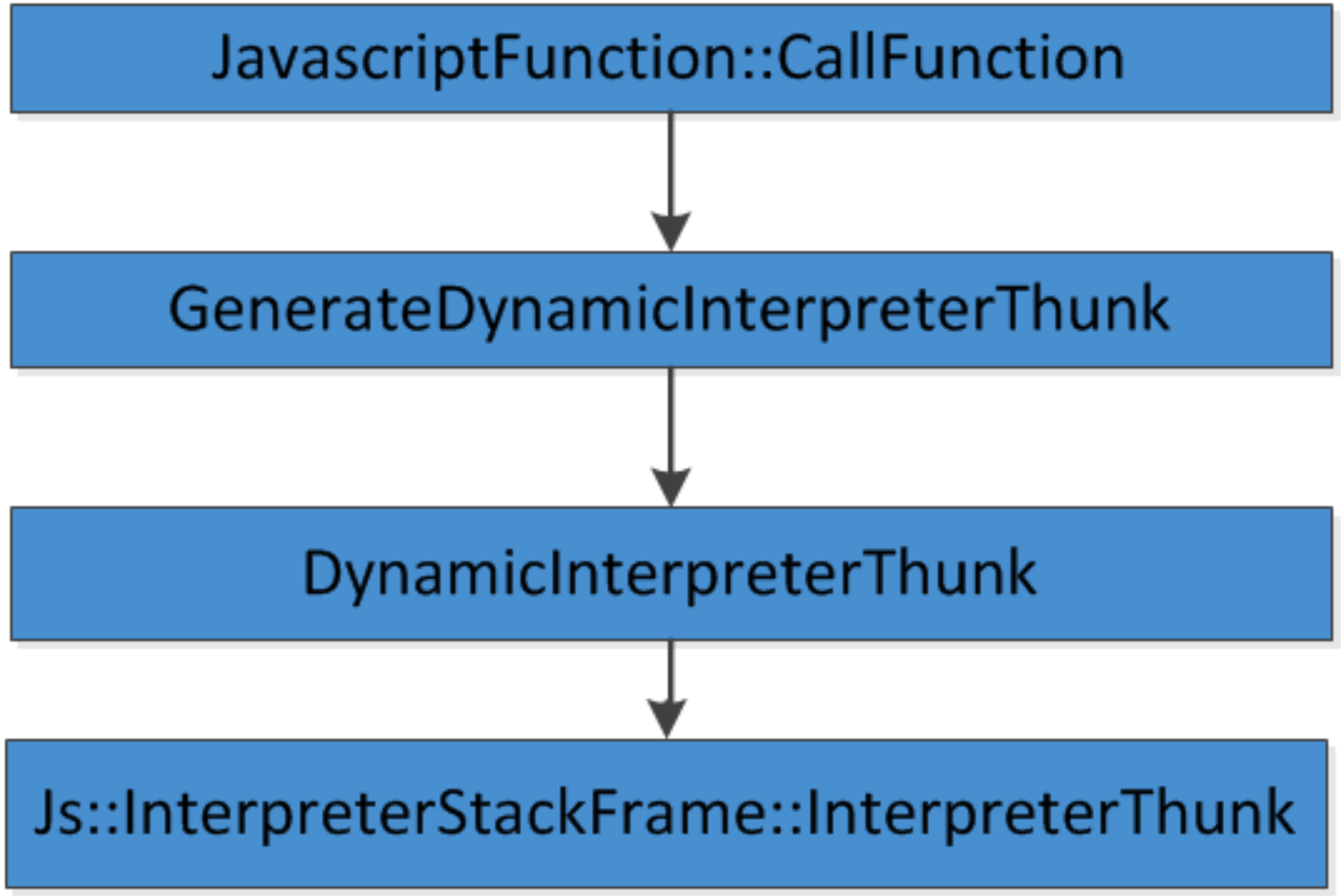
pCodeStorageSection

fake CodeStorageSection
0x0c shellcode_address
0x10 size(equal to 0xd00)

Vuln 3

- **Use InterpreterThunkEmitter to bypass CFG**
- **Vuln Type: No Control Flow Guard check**
- **Module: chakra.dll**
- **Operation System: Windows 10 14328 32 bit**
- **Bypass CFG**

Vuln 3:Js Function Interpreting Execute



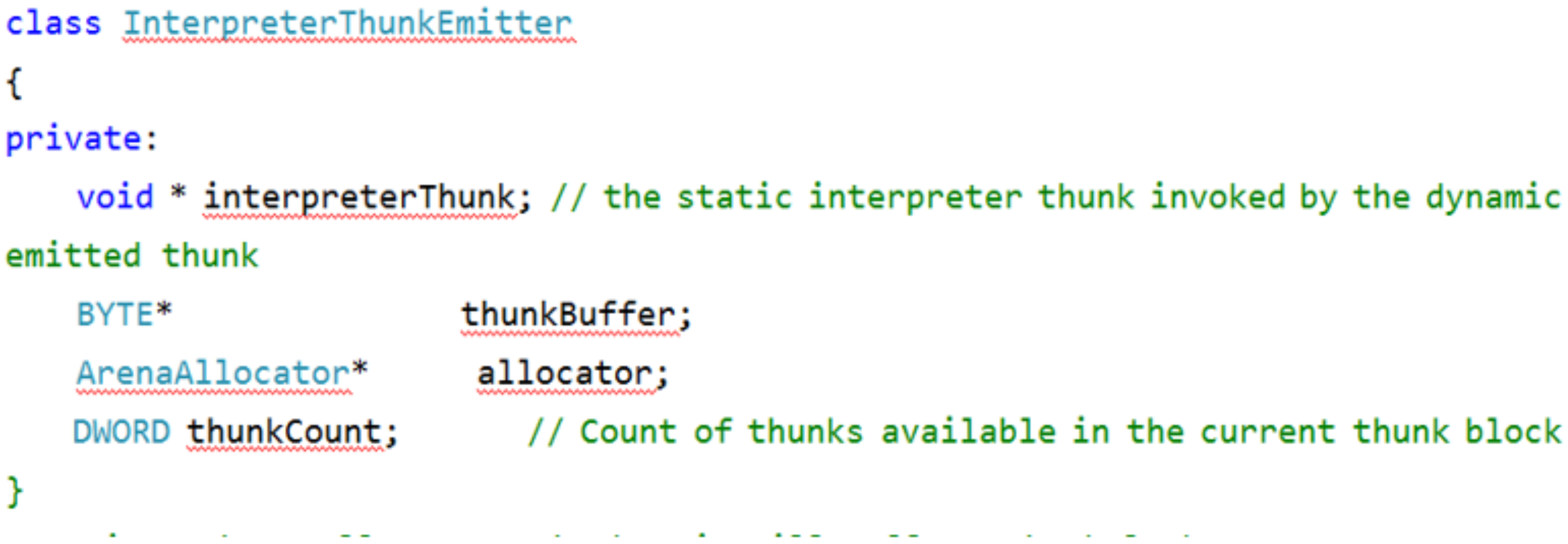
JavascriptFunction::CallFunction

GenerateDynamicInterpreterThunk

DynamicInterpreterThunk

Js::InterpreterStackFrame::InterpreterThunk

Vuln 3: InterpreterThunkEmitter



```
class InterpreterThunkEmitter
{
private:
    void * interpreterThunk; // the static interpreter thunk invoked by the dynamic
emitted thunk
    BYTE*                 thunkBuffer;
    ArenaAllocator*        allocator;
    DWORD thunkCount;          // Count of thunks available in the current thunk block
}
```

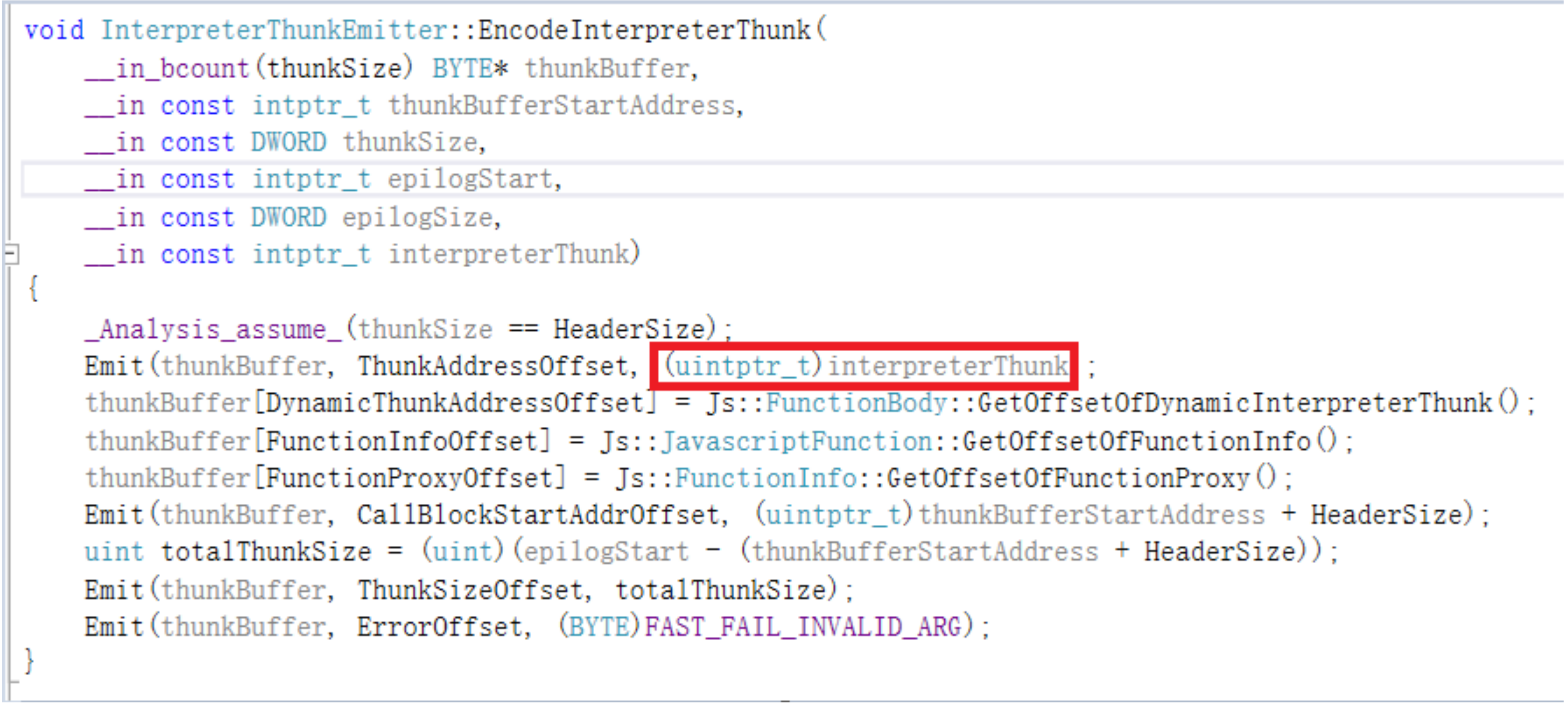
Vuln 3

```
BYTE* InterpreterThunkEmitter::GetNextThunk(PVOID* ppDynamicInterpreterThunk)
{
    Assert(ppDynamicInterpreterThunk);
    Assert(*ppDynamicInterpreterThunk == nullptr);

    if(thunkCount == 0)
    {
        if(!this->freeListedThunkBlocks.Empty())
        {
            return AllocateFromFreeList(ppDynamicInterpreterThunk);
        }
        NewThunkBlock();
    }
```

Vuln 3

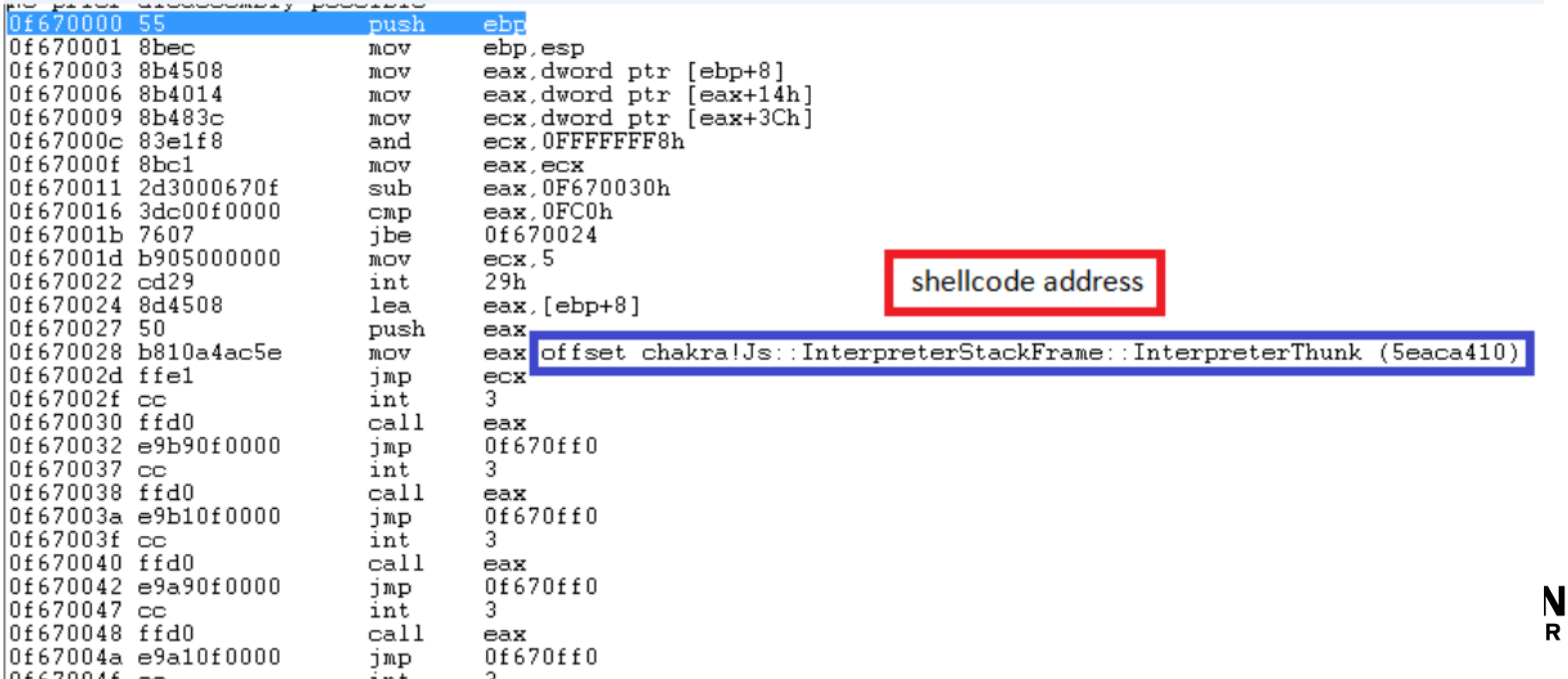
```
const BYTE InterpreterThunkEmitter::InterpreterThunk[] = {
    0x55,                           //  push      ebp              ;Prolog - setup the stack frame
    0x8B, 0xEC,                     //  mov       ebp,esp
    0x8B, 0x45, 0x08,               //  mov       eax, dword ptr [ebp+8]
    0x8B, 0x40, 0x00,               //  mov       eax, dword ptr [eax+FunctionBodyOffset]
    0x8B, 0x48, 0x00,               //  mov       ecx, dword ptr [eax+DynamicThunkAddressOffset]
                                    //  Range Check for Valid call target
    0x83, 0xE1, 0xF8,               //  and       ecx, 0FFFFFFF8h
    0x8b, 0xc1,                     //  mov       eax, ecx
    0x2d, 0x00, 0x00, 0x00, 0x00,   //  sub       eax, CallBlockStartAddress
    0x3d, 0x00, 0x00, 0x00, 0x00,   //  cmp       eax, ThunkSize
    0x76, 0x07,                     //  jbe       SHORT $safe
    0xb9, 0x00, 0x00, 0x00, 0x00,   //  mov       ecx, errorcode
    0xCD, 0x29,                     //  int       29h
    //$safe
    0x8D, 0x45, 0x08,               //  lea       eax, ebp+8
    0x50,                           //  push      eax
    0xB8, 0x00, 0x00, 0x00, 0x00,   //  mov       eax, <thunk>//static InterpreterThunk address
    0xFF, 0xE1,                     //  jmp       ecx
    0xCC                            //  int 3 for 8byte alignment
```

Vuln 3:Set Dynamic InterpreterThunk Address



```
void InterpreterThunkEmitter::EncodeInterpreterThunk(
    __in_bcount(thunkSize) BYTE* thunkBuffer,
    __in const intptr_t thunkBufferStartAddress,
    __in const DWORD thunkSize,
    __in const intptr_t epilogStart,
    __in const DWORD epilogSize,
    __in const intptr_t interpreterThunk)
{
    _Analysis_assume_(thunkSize == HeaderSize);
    Emit(thunkBuffer, ThunkAddressOffset, (uintptr_t)interpreterThunk);
    thunkBuffer[DynamicThunkAddressOffset] = Js::FunctionBody::GetOffsetOfDynamicInterpreterThunk();
    thunkBuffer[FunctionInfoOffset] = Js::JavascriptFunction::GetOffsetOfFunctionInfo();
    thunkBuffer[FunctionProxyOffset] = Js::FunctionInfo::GetOffsetOfFunctionProxy();
    Emit(thunkBuffer, CallBlockStartAddrOffset, (uintptr_t)thunkBufferStartAddress + HeaderSize);
    uint totalThunkSize = (uint)(epilogStart - (thunkBufferStartAddress + HeaderSize));
    Emit(thunkBuffer, ThunkSizeOffset, totalThunkSize);
    Emit(thunkBuffer, ErrorOffset, (BYTE)FAST_FAIL_INVALID_ARG);
}
```

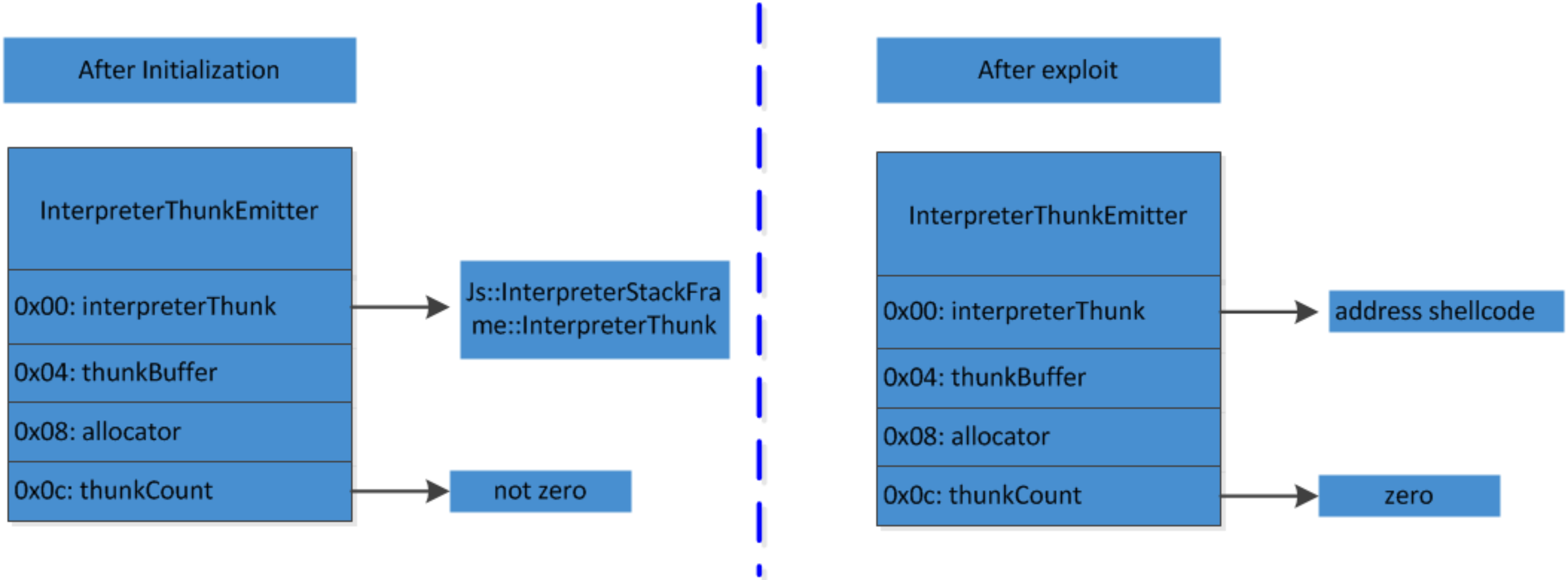
Vuln 3:Dynamic InterpreterThunk



```
No prior disassembly possible
0f670000 55              push    ebp
0f670001 8bec            mov     ebp,esp
0f670003 8b4508          mov     eax,dword ptr [ebp+8]
0f670006 8b4014          mov     eax,dword ptr [eax+14h]
0f670009 8b483c          mov     ecx,dword ptr [eax+3Ch]
0f67000c 83e1f8          and     ecx,0FFFFFFF8h
0f67000f 8bc1            mov     eax,ecx
0f670011 2d3000670f      sub     eax,0F670030h
0f670016 3dc00f0000      cmp     eax,0FC0h
0f67001b 7607            jbe     0f670024
0f67001d b905000000      mov     ecx,5
0f670022 cd29            int     29h
0f670024 8d4508          lea     eax,[ebp+8]
0f670027 50              push    eax
0f670028 b810a4ac5e      mov     eax offset chakra!Js::InterpreterStackFrame::InterpreterThunk (5eaca410)
0f67002d ffe1            jmp     ecx
0f67002f cc              int     3
0f670030 ffd0            call    eax
0f670032 e9b90f0000      jmp     0f670ff0
0f670037 cc              int     3
0f670038 ffd0            call    eax
0f67003a e9b10f0000      jmp     0f670ff0
0f67003f cc              int     3
0f670040 ffd0            call    eax
0f670042 e9a90f0000      jmp     0f670ff0
0f670047 cc              int     3
0f670048 ffd0            call    eax
0f67004a e9a10f0000      jmp     0f670ff0
0f67004f cc              int     3
```

shellcode address

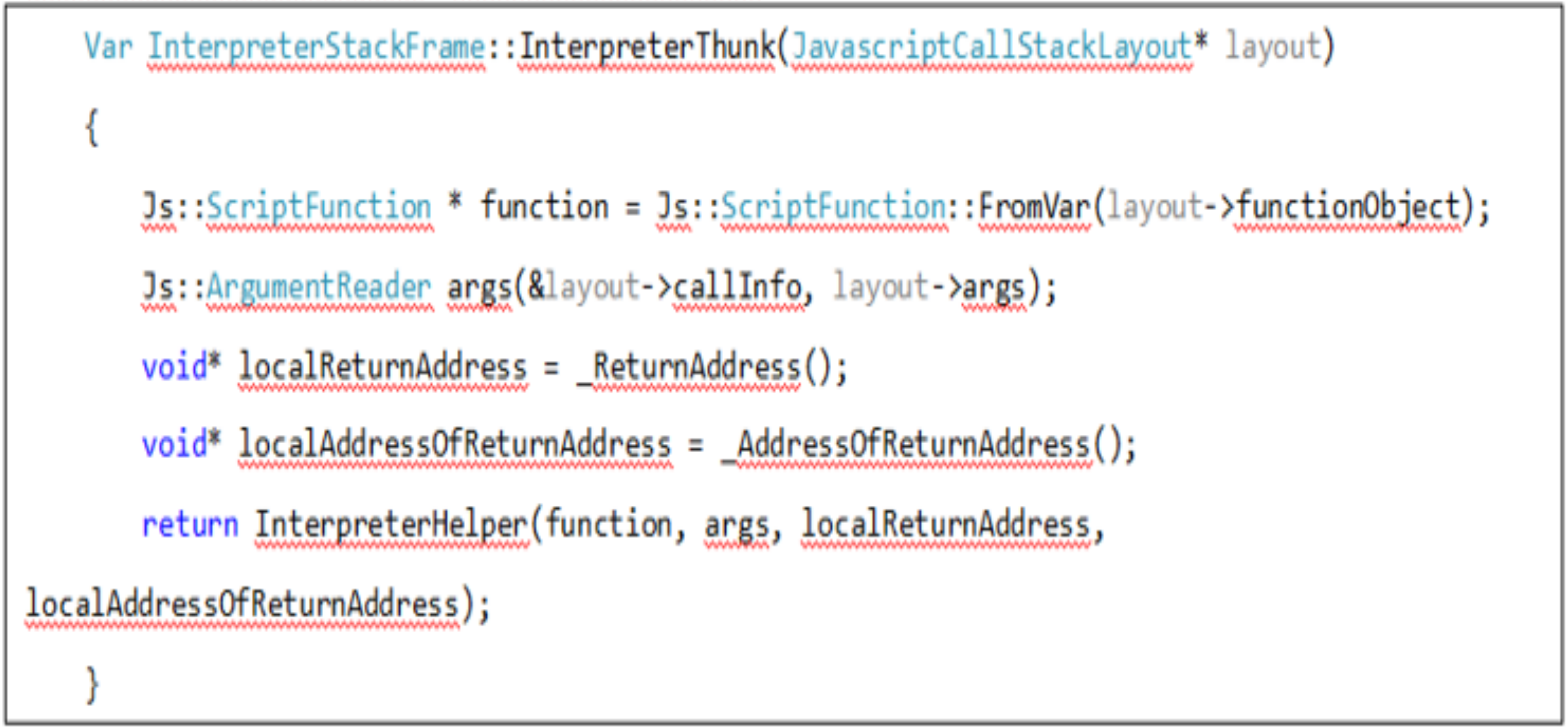
Vuln 3: Exploit



Vuln 4

- **Write the return address to bypass CFG and DEP**
- **Vuln Type: write return address**
- **Module: chakra.dll**
- **Operation System: Windows 10 14352 32 bit**
- **BYPASS CFG/RFG**

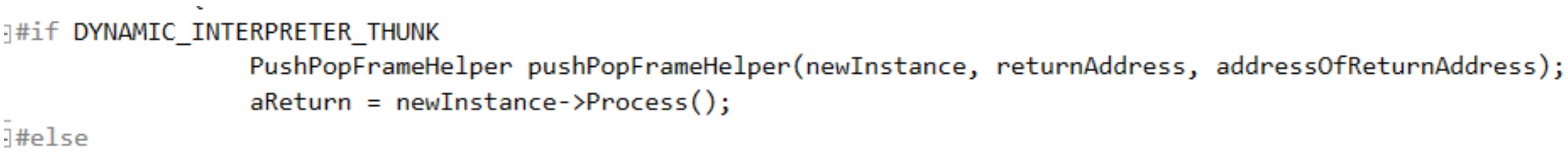
Vuln 4



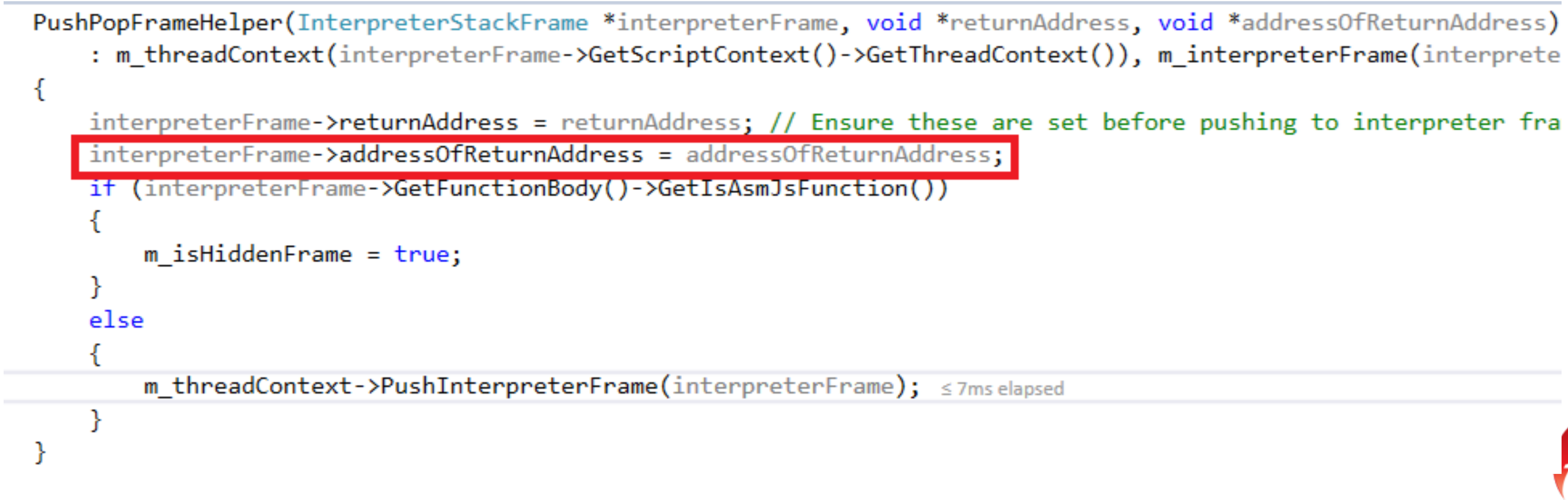
```
    Var InterpreterStackFrame::InterpreterThunk(JavascriptCallStackLayout* layout)
    {
        Js::ScriptFunction * function = Js::ScriptFunction::FromVar(layout->functionObject);
        Js::ArgumentReader args(&layout->callInfo, layout->args);
        void* localReturnAddress = _ReturnAddress();
        void* localAddressOfReturnAddress = _AddressOfReturnAddress();
        return InterpreterHelper(function, args, localReturnAddress,
localAddressOfReturnAddress);
    }
```

Vuln 4

- **InterpreterHelper will call following function**



```
#if DYNAMIC_INTERPRETER_THUNK
                PushPopFrameHelper pushPopFrameHelper(newInstance, returnAddress, addressOfReturnAddress);
                aReturn = newInstance->Process();
#else
```

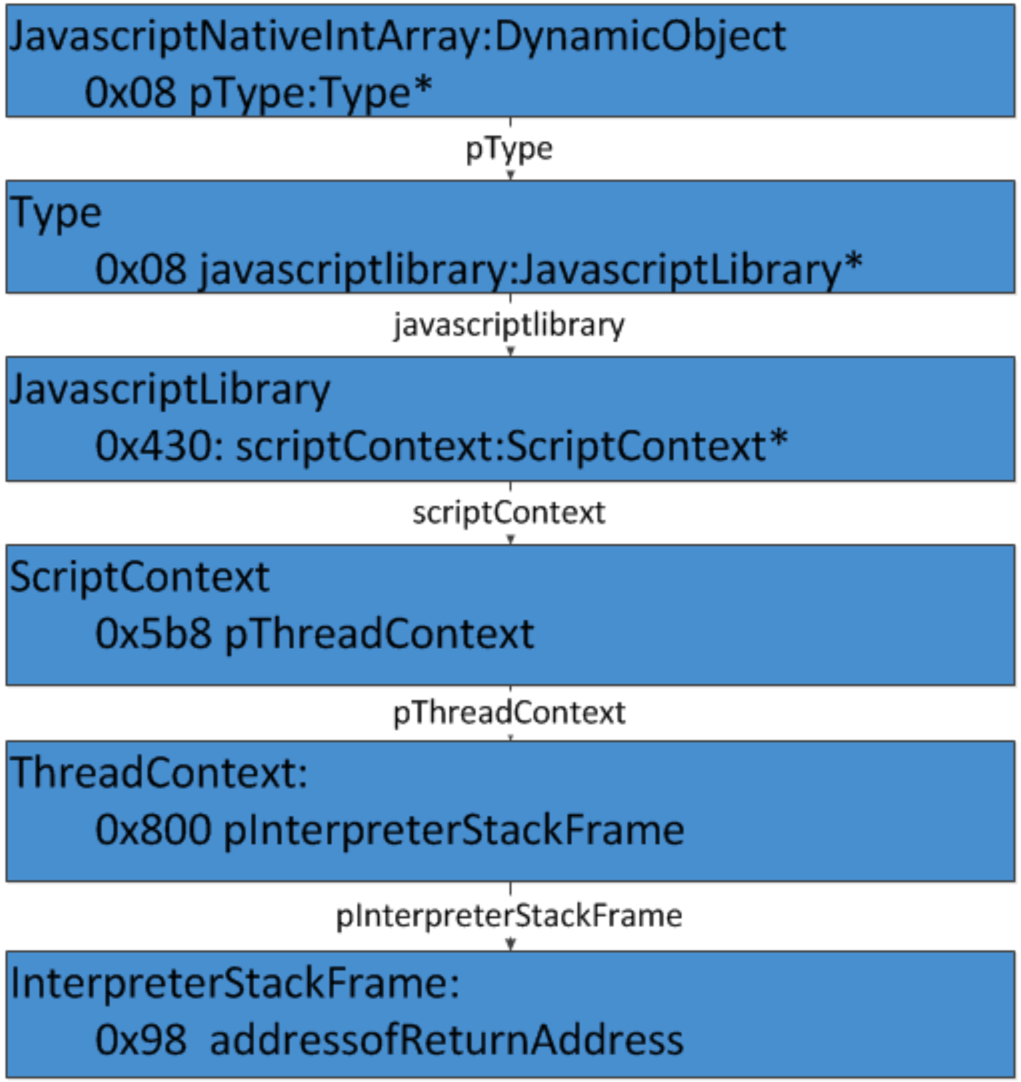


```
 PushPopFrameHelper(InterpreterStackFrame *interpreterFrame, void *returnAddress, void *addressOfReturnAddress)
     : m_threadContext(interpreterFrame->GetScriptContext()->GetThreadContext()), m_interpreterFrame(interprete
 {
     interpreterFrame->returnAddress = returnAddress; // Ensure these are set before pushing to interpreter fra
     interpreterFrame->addressOfReturnAddress = addressOfReturnAddress;
     if (interpreterFrame->GetFunctionBody()->GetIsAsmJsFunction())
     {
         m_isHiddenFrame = true;
     }
     else
     {
         m_threadContext->PushInterpreterFrame(interpreterFrame);  ≤ 7ms elapsed
     }
 }
```

Vuln 4

- **InterpreterStackFrame**
  - **0x48 addressOfReturnAddress**

Vuln 4: Exploit Method



JavascriptNativeIntArray:DynamicObject
    0x08 pType:Type*

pType

Type
    0x08 javascriptlibrary:JavascriptLibrary*

javascriptlibrary

JavascriptLibrary
    0x430: scriptContext:ScriptContext*

scriptContext

ScriptContext
    0x5b8 pThreadContext

pThreadContext

ThreadContext:
    0x800 pInterpreterStackFrame

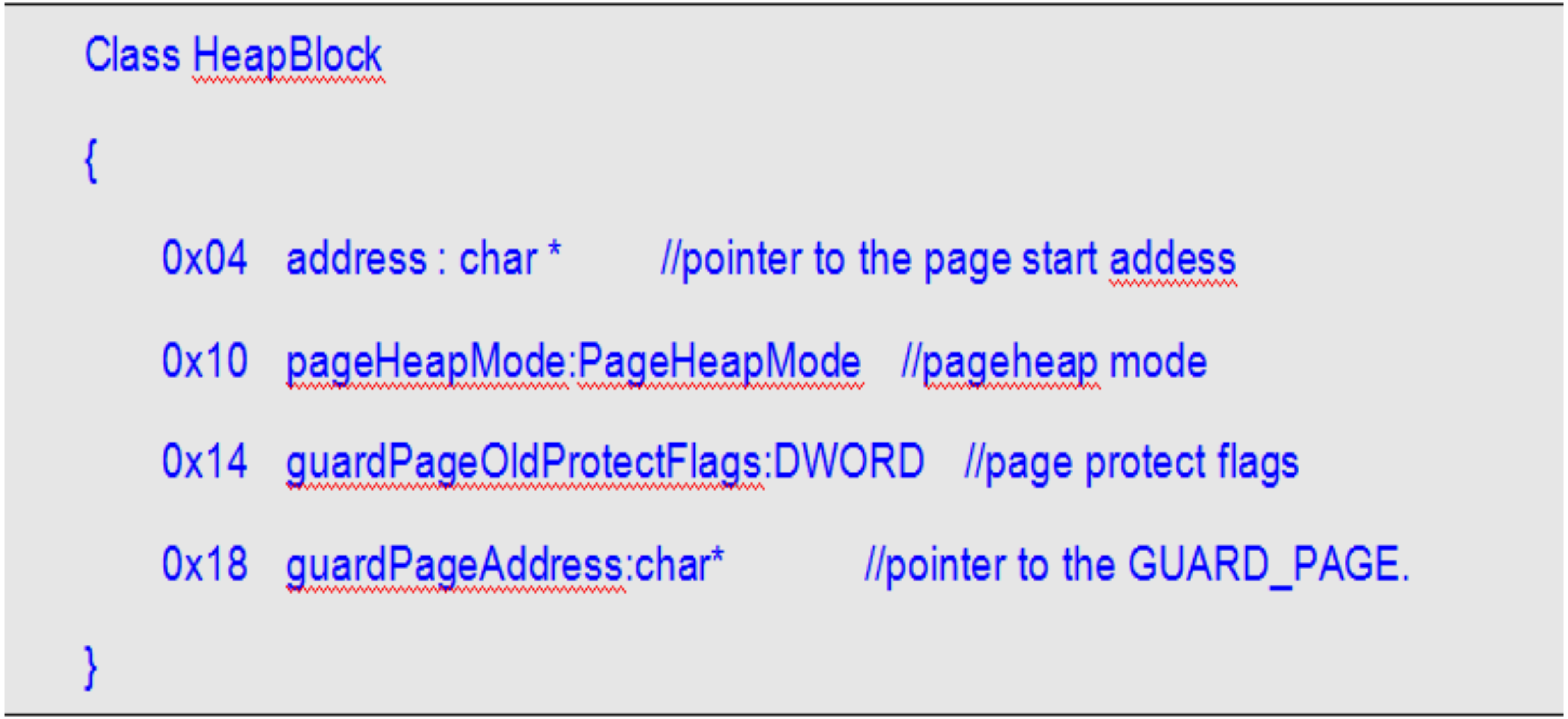
pInterpreterStackFrame

InterpreterStackFrame:
    0x98 addressofReturnAddress

Vuln 5

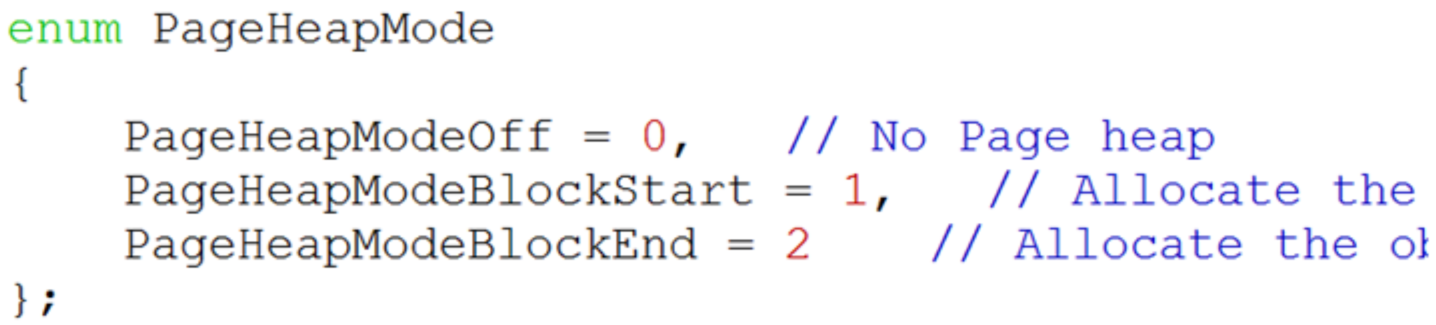
- **Use Chakra Recycler Memory pageheap to bypass DEP and CFG**
- **Vuln type: Data Only Attack**
- **Module: chakra.dll**
- **Operation System: Windows 10 14328 32 bit**
- **BYPASS CFG/DEP**

Vuln 5

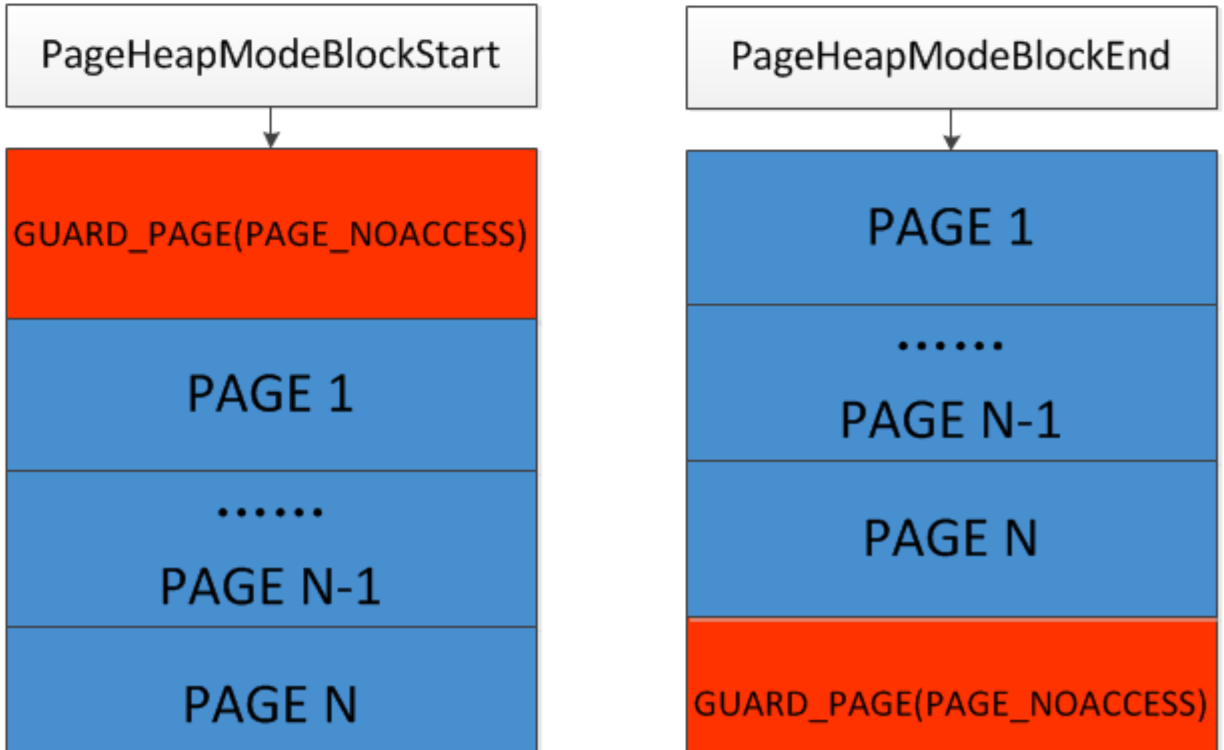


```
Class HeapBlock

{

        0x04   address : char *          //pointer to the page start addess

        0x10   pageHeapMode:PageHeapMode    //pageheap mode

        0x14   guardPageOldProtectFlags:DWORD    //page protect flags

        0x18   guardPageAddress:char*           //pointer to the GUARD_PAGE.

}
```

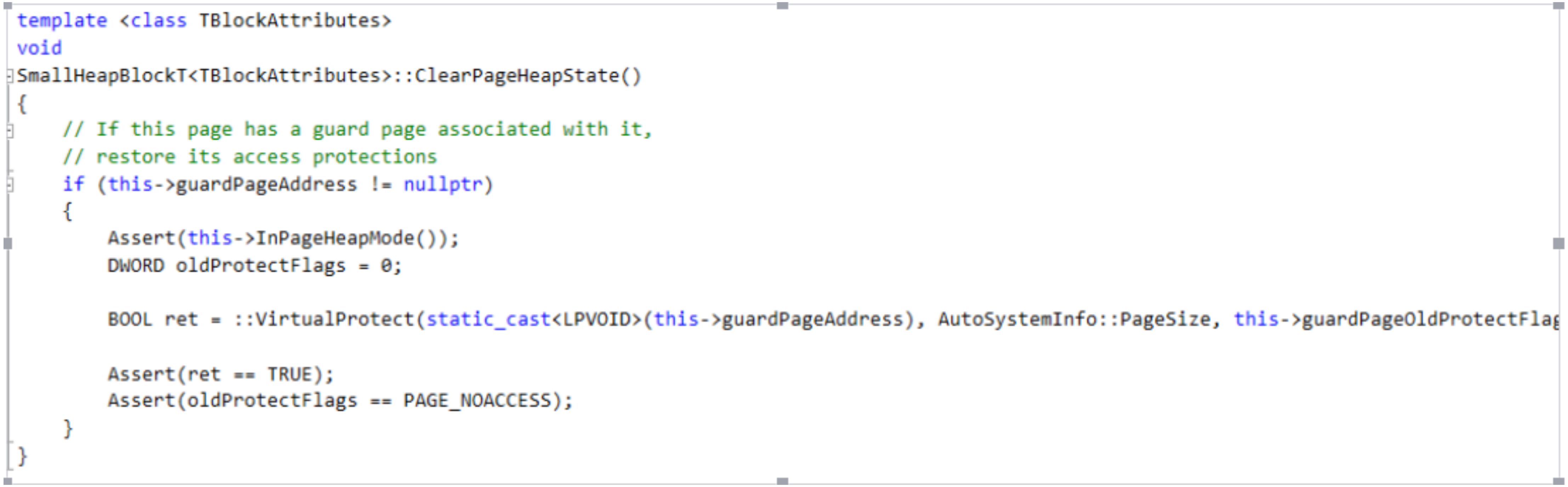
Vuln 5



```
enum PageHeapMode
{
    PageHeapModeOff = 0,    // No Page heap
    PageHeapModeBlockStart = 1,    // Allocate the
    PageHeapModeBlockEnd = 2    // Allocate the ol
};
```

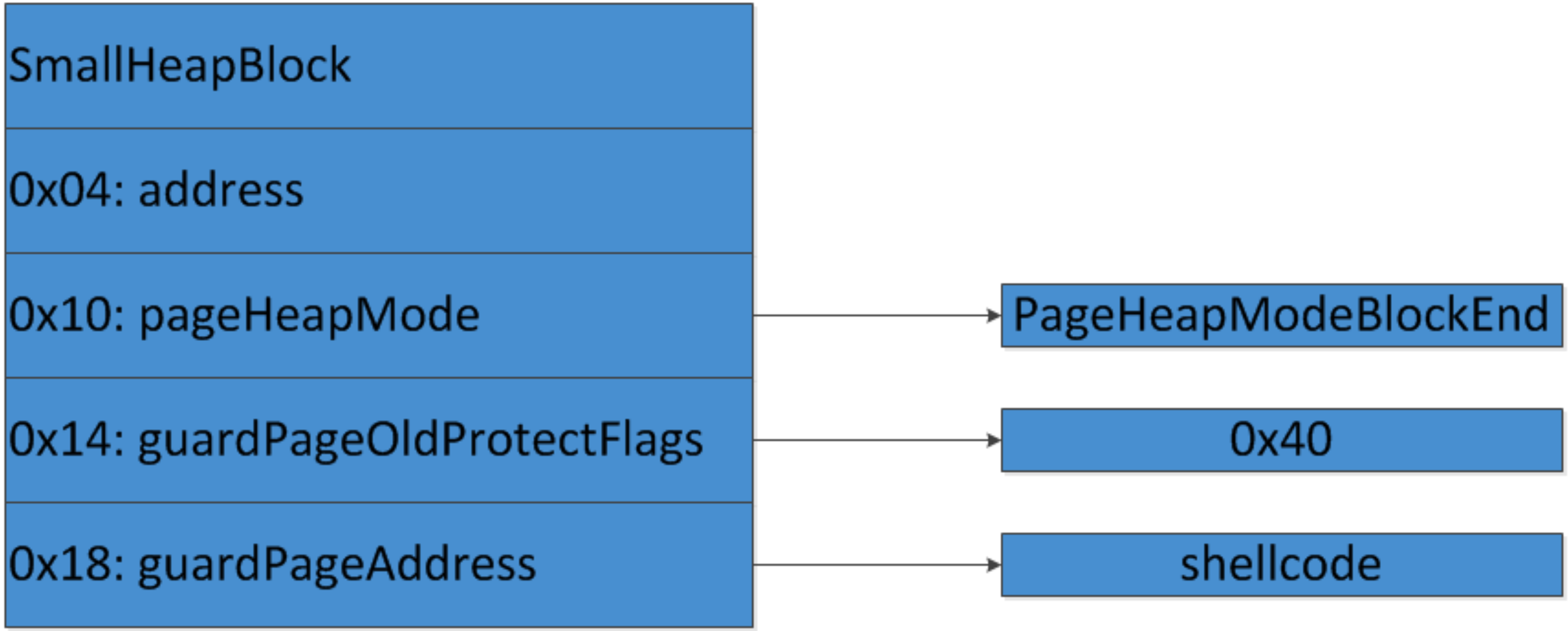


Vuln 5



```
template <class TBlockAttributes>
void
SmallHeapBlockT<TBlockAttributes>::ClearPageHeapState()
{
    // If this page has a guard page associated with it,
    // restore its access protections
    if (this->guardPageAddress != nullptr)
    {
        Assert(this->InPageHeapMode());
        DWORD oldProtectFlags = 0;

        BOOL ret = ::VirtualProtect(static_cast<LPVOID>(this->guardPageAddress), AutoSystemInfo::PageSize, this->guardPageOldProtectFlag

        Assert(ret == TRUE);
        Assert(oldProtectFlags == PAGE_NOACCESS);
    }
}
```

Vuln 5:Exploit Method



SmallHeapBlock

0x04: address

0x10: pageHeapMode → PageHeapModeBlockEnd

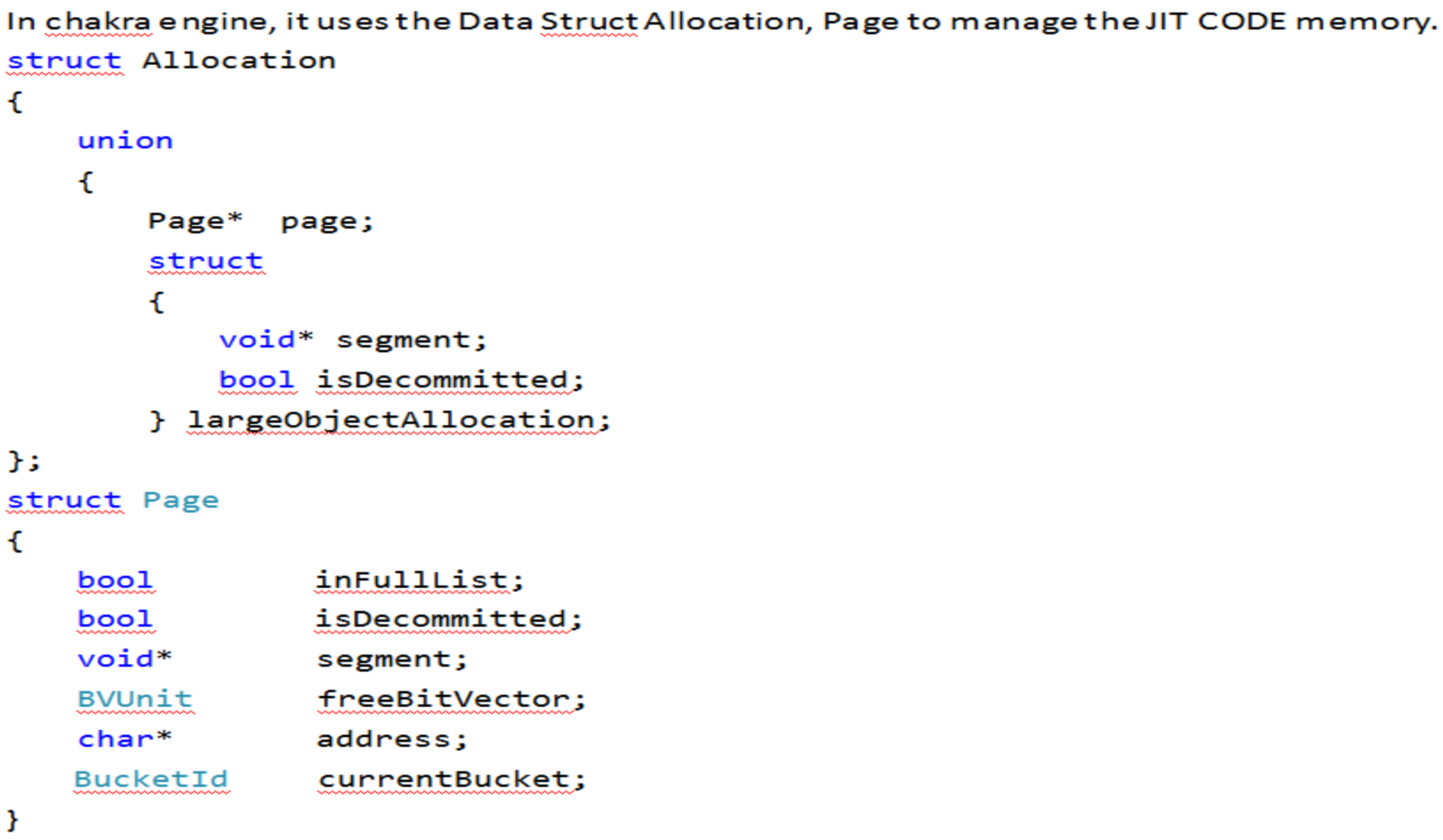
0x14: guardPageOldProtectFlags → 0x40

0x18: guardPageAddress → shellcode

Vuln 6

- **Use JIT PAGE to bypass CFG and DEP**
- **Vuln Type: Data Only Attack**
- **Module: chakra.dll**
- **Operation System: Windows 10 14361 32 bit**
- **BYPASS CFG/DEP**

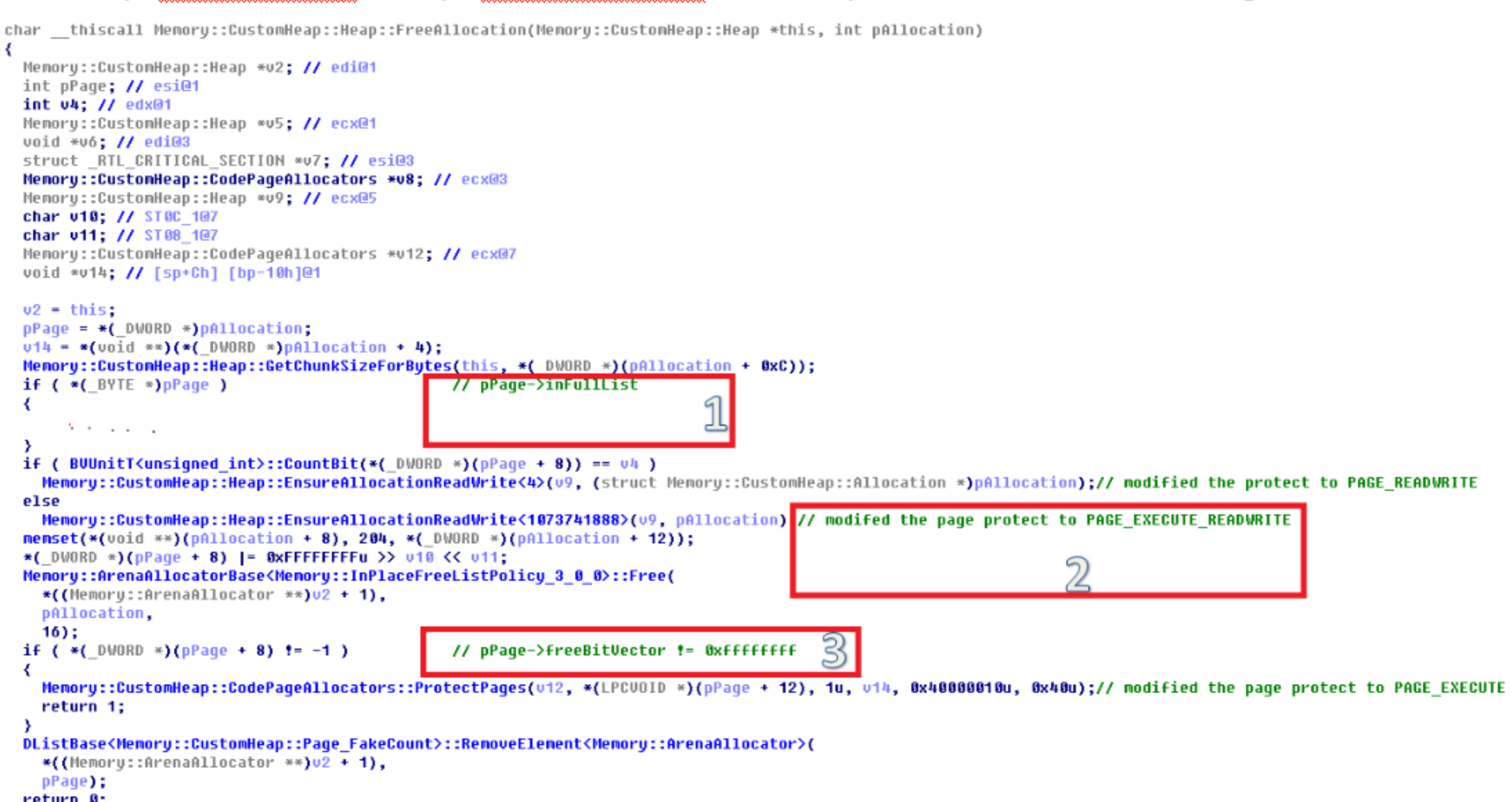
Vuln 6



In chakra engine, it uses the Data Struct Allocation, Page to manage the JIT CODE memory.

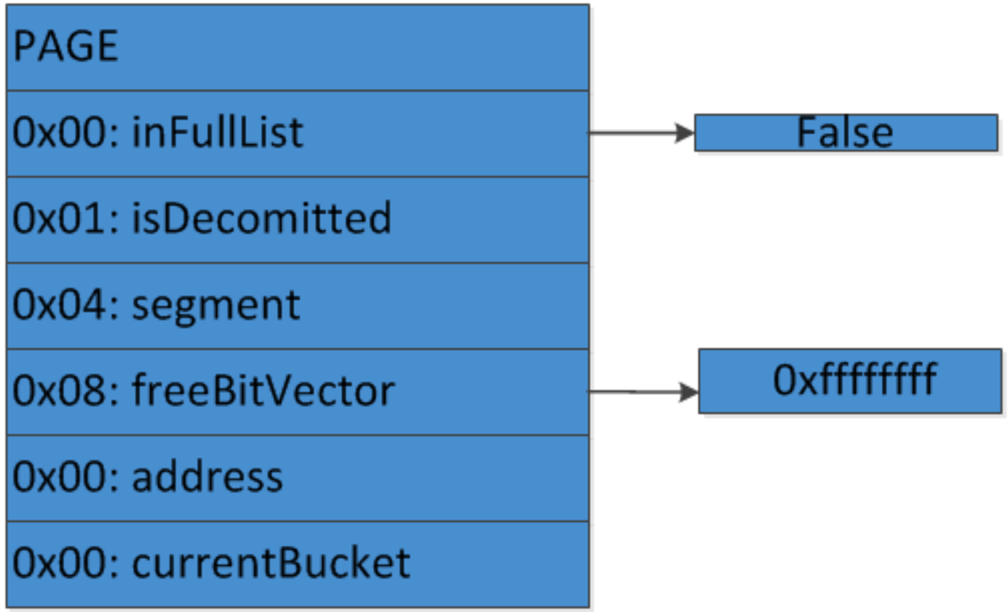
```
struct Allocation
{
    union
    {
        Page*  page;
        struct
        {
            void* segment;
            bool isDecommitted;
        } largeObjectAllocation;
};
struct Page
{
    bool            inFullList;
    bool            isDecommitted;
    void*           segment;
    BVUnit          freeBitVector;
    char*           address;
    BucketId        currentBucket;
}
```

Vuln 6



```
char __thiscall Memory::CustomHeap::Heap::FreeAllocation(Memory::CustomHeap::Heap *this, int pAllocation)
{
  Memory::CustomHeap::Heap *v2; // edi@1
  int pPage; // esi@1
  int v4; // edx@1
  Memory::CustomHeap::Heap *v5; // ecx@1
  void *v6; // edi@3
  struct _RTL_CRITICAL_SECTION *v7; // esi@3
  Memory::CustomHeap::CodePageAllocators *v8; // ecx@3
  Memory::CustomHeap::Heap *v9; // ecx@5
  char v10; // ST0C_1@7
  char v11; // ST08_1@7
  Memory::CustomHeap::CodePageAllocators *v12; // ecx@7
  void *v14; // [sp+Ch] [bp-10h]@1

  v2 = this;
  pPage = *(_DWORD *)pAllocation;
  v14 = *(void **)(*(_DWORD *)pAllocation + 4);
  Memory::CustomHeap::Heap::GetChunkSizeForBytes(this, *(_DWORD *)(pAllocation + 0xC));
  if ( *(_BYTE *)pPage )                         // pPage->inFullList
  {
      . .  . .  .
  }
  if ( BVUnitT<unsigned_int>::CountBit(*(_DWORD *)(pPage + 8)) == v4 )
    Memory::CustomHeap::Heap::EnsureAllocationReadWrite<4>(v9, (struct Memory::CustomHeap::Allocation *)pAllocation);// modified the protect to PAGE_READWRITE
  else
    Memory::CustomHeap::Heap::EnsureAllocationReadWrite<1073741888>(v9, pAllocation) // modifed the page protect to PAGE_EXECUTE_READWRITE
  memset(*(void **)(pAllocation + 8), 204, *(_DWORD *)(pAllocation + 12));
  *(_DWORD *)(pPage + 8) |= 0xFFFFFFFFu >> v10 << v11;
  Memory::ArenaAllocatorBase<Memory::InPlaceFreeListPolicy_3_0_0>::Free(
    *((Memory::ArenaAllocator **)v2 + 1),
    pAllocation,
    16);
  if ( *(_DWORD *)(pPage + 8) != -1 )            // pPage->freeBitVector != 0xffffffff
  {
    Memory::CustomHeap::CodePageAllocators::ProtectPages(v12, *(LPCVOID *)(pPage + 12), 1u, v14, 0x40000010u, 0x40u);// modified the page protect to PAGE_EXECUTE
    return 1;
  }
  DListBase<Memory::CustomHeap::Page_FakeCount>::RemoveElement<Memory::ArenaAllocator>(
    *((Memory::ArenaAllocator **)v2 + 1),
    pPage);
  return 0;
```

Vuln 6:Exploit Method



PAGE

0x00: inFullList

0x01: isDecomitted

0x04: segment

0x08: freeBitVector

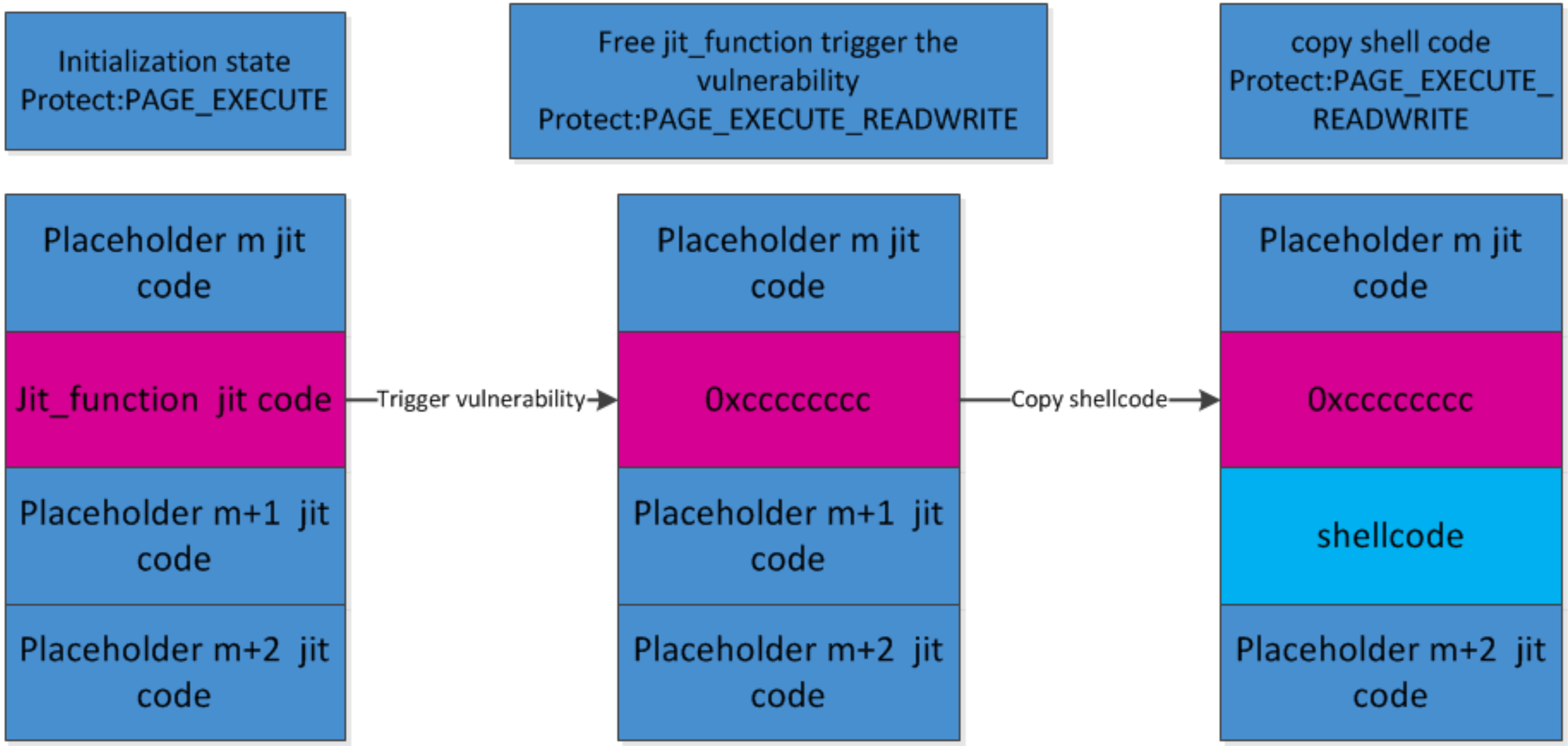
0x00: address

0x00: currentBucket

False

0xffffffff

Vuln 6:Exploit Method



Initialization state
Protect:PAGE_EXECUTE

Free jit_function trigger the vulnerability
Protect:PAGE_EXECUTE_READWRITE

copy shell code
Protect:PAGE_EXECUTE_READWRITE

Placeholder m jit code
Jit_function jit code
Placeholder m+1 jit code
Placeholder m+2 jit code

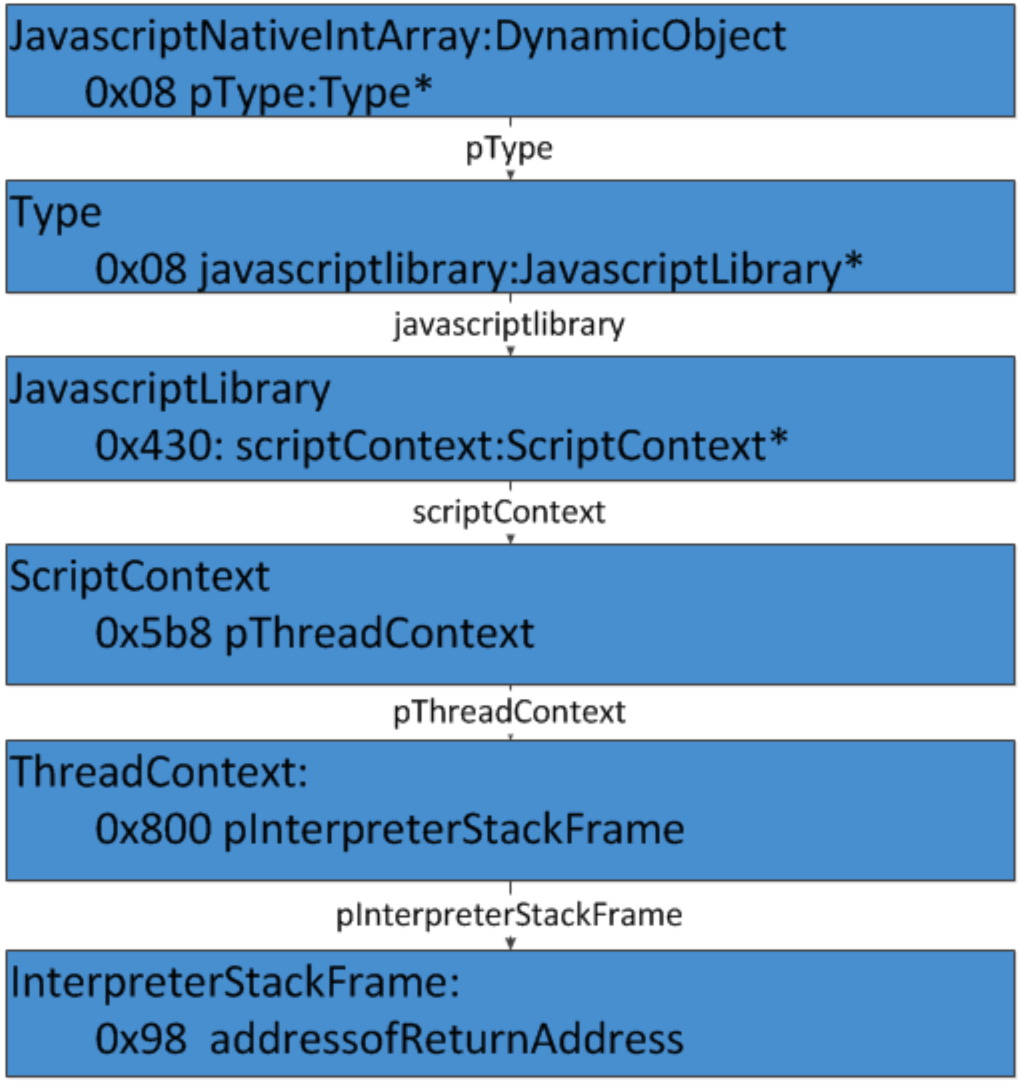
Placeholder m jit code
0xcccccccc
Placeholder m+1 jit code
Placeholder m+2 jit code

Placeholder m jit code
0xcccccccc
shellcode
Placeholder m+2 jit code

Exploit Framework

- Write Return Address
- VirtualAlloc/VirtualProtect

Exploit Vuln 4:Get addressofReturnAddress



JavascriptNativeIntArray:DynamicObject
0x08 pType:Type*

pType

Type
0x08 javascriptlibrary:JavascriptLibrary*

javascriptlibrary

JavascriptLibrary
0x430: scriptContext:ScriptContext*

scriptContext

ScriptContext
0x5b8 pThreadContext

pThreadContext

ThreadContext:
0x800 pInterpreterStackFrame

pInterpreterStackFrame

InterpreterStackFrame:
0x98 addressofReturnAddress

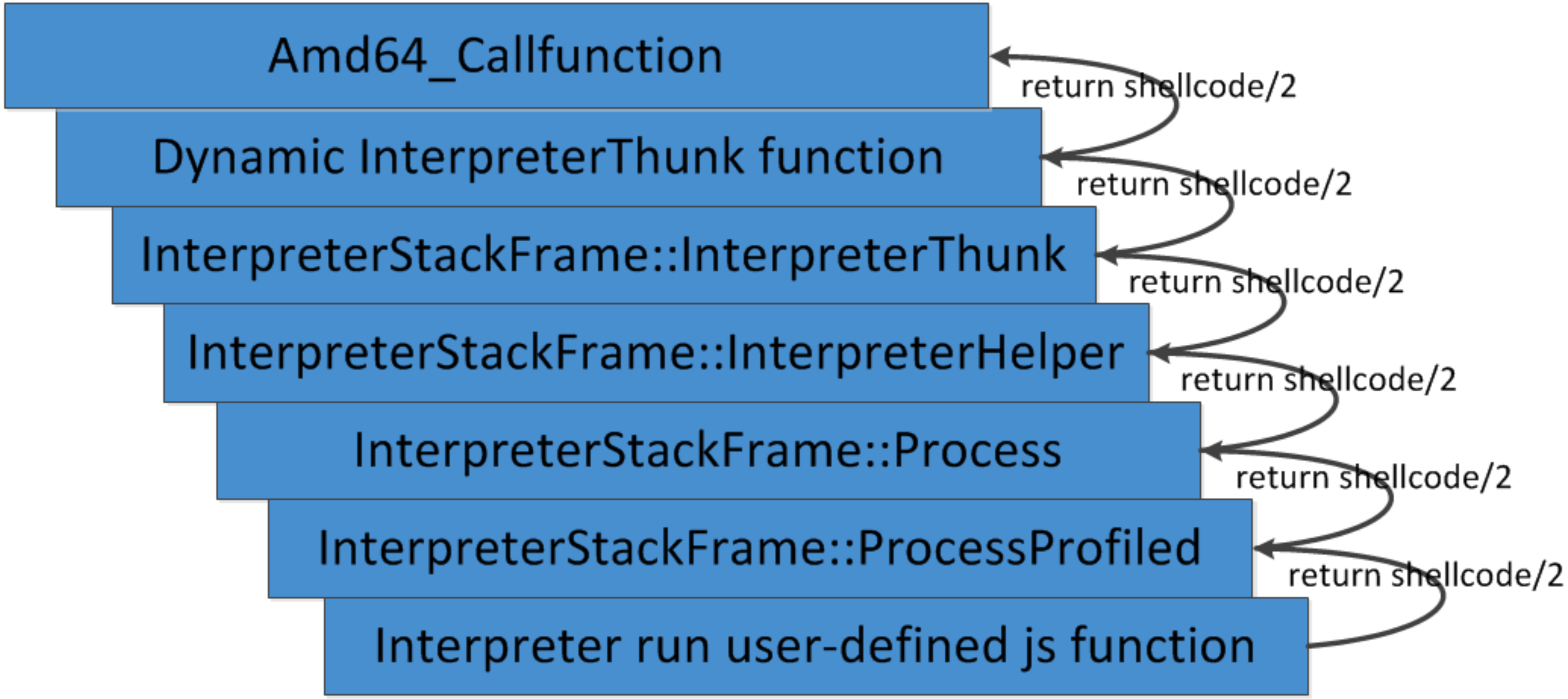
Exploit Vuln 4

What to write in the addressOfReturnAddress?

Shellcode address?

Stack pivot address xchg eax,esp

Interpreter CallStack



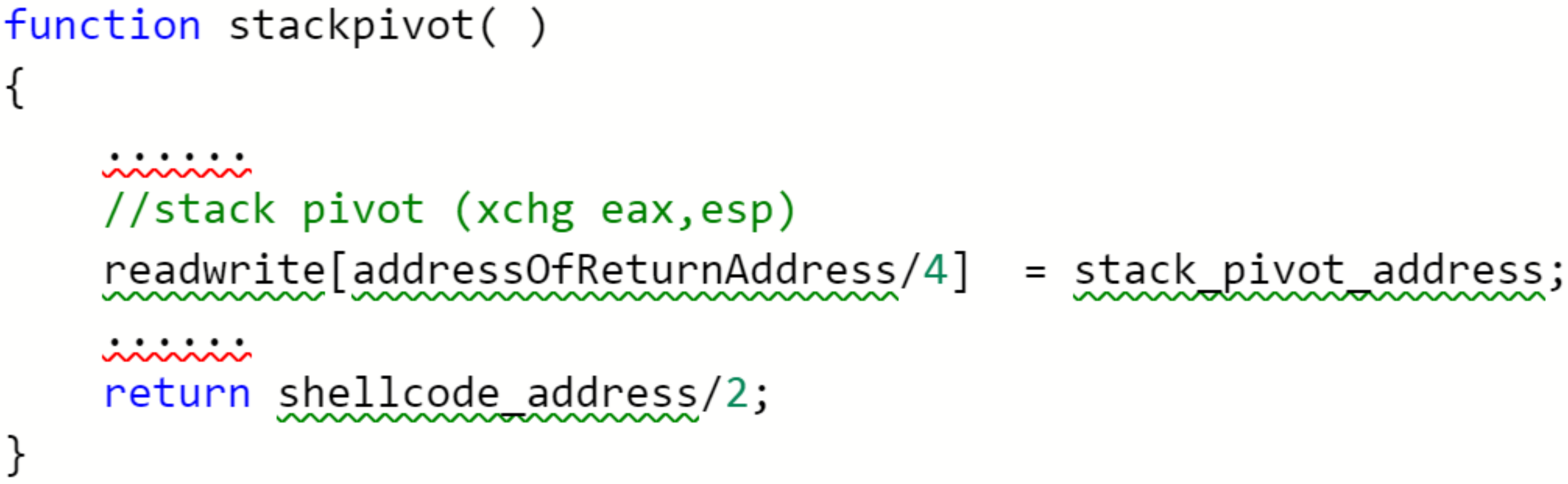
Amd64_Callfunction
Dynamic InterpreterThunk function
InterpreterStackFrame::InterpreterThunk
InterpreterStackFrame::InterpreterHelper
InterpreterStackFrame::Process
InterpreterStackFrame::ProcessProfiled
Interpreter run user-defined js function
return shellcode/2
return shellcode/2
return shellcode/2
return shellcode/2
return shellcode/2
return shellcode/2

Exploit Vuln 4:stackpivot function

Construct a function, I call it StackPivot,do two things:

I. write the stack pivot gadget address to the return address

II.Return shellcode_address/2



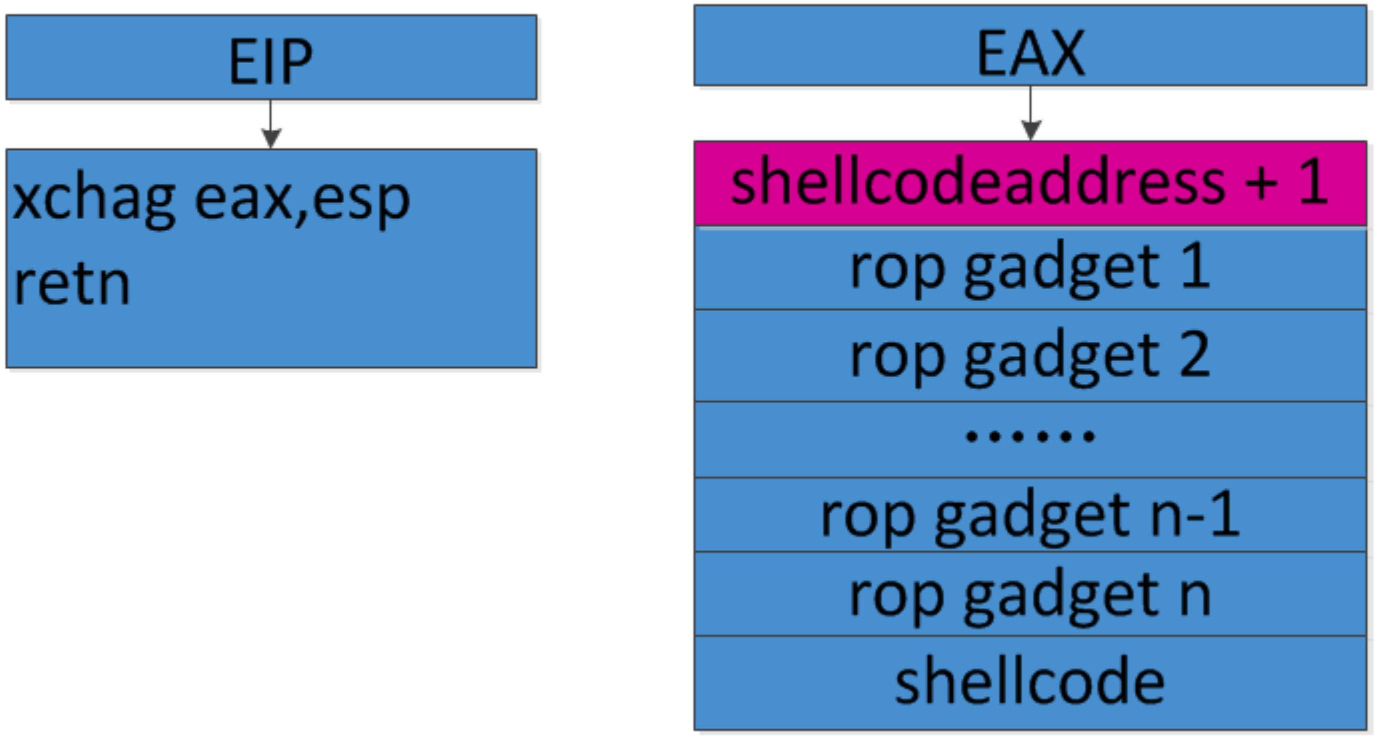
```
function stackpivot( )
{
    .......
    //stack pivot (xchg eax,esp)
    readwrite[addressOfReturnAddress/4]  = stack_pivot_address;
    .......
    return shellcode_address/2;
}
```

Exploit Vuln 4:stackpivot function

- **The representation of an integer in memory(on x86)**
  - In chakra engine, script defined an integer is m, in memory it’s 2*m + 1

Exploit Vuln 4: Stackpivot function

First:Static interpreterThunk return



EIP

xchag eax,esp
retn

EAX

shellcodeaddress + 1

rop gadget 1

rop gadget 2

·······

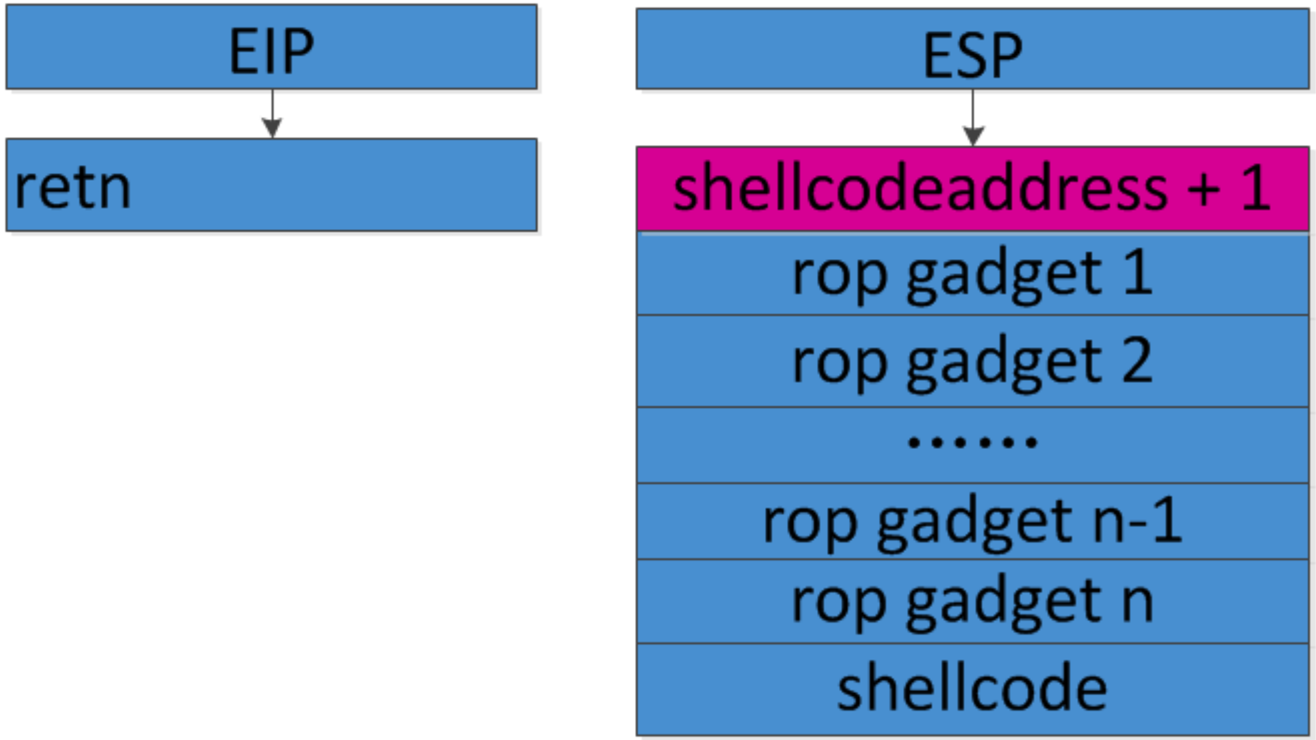
rop gadget n-1

rop gadget n

shellcode

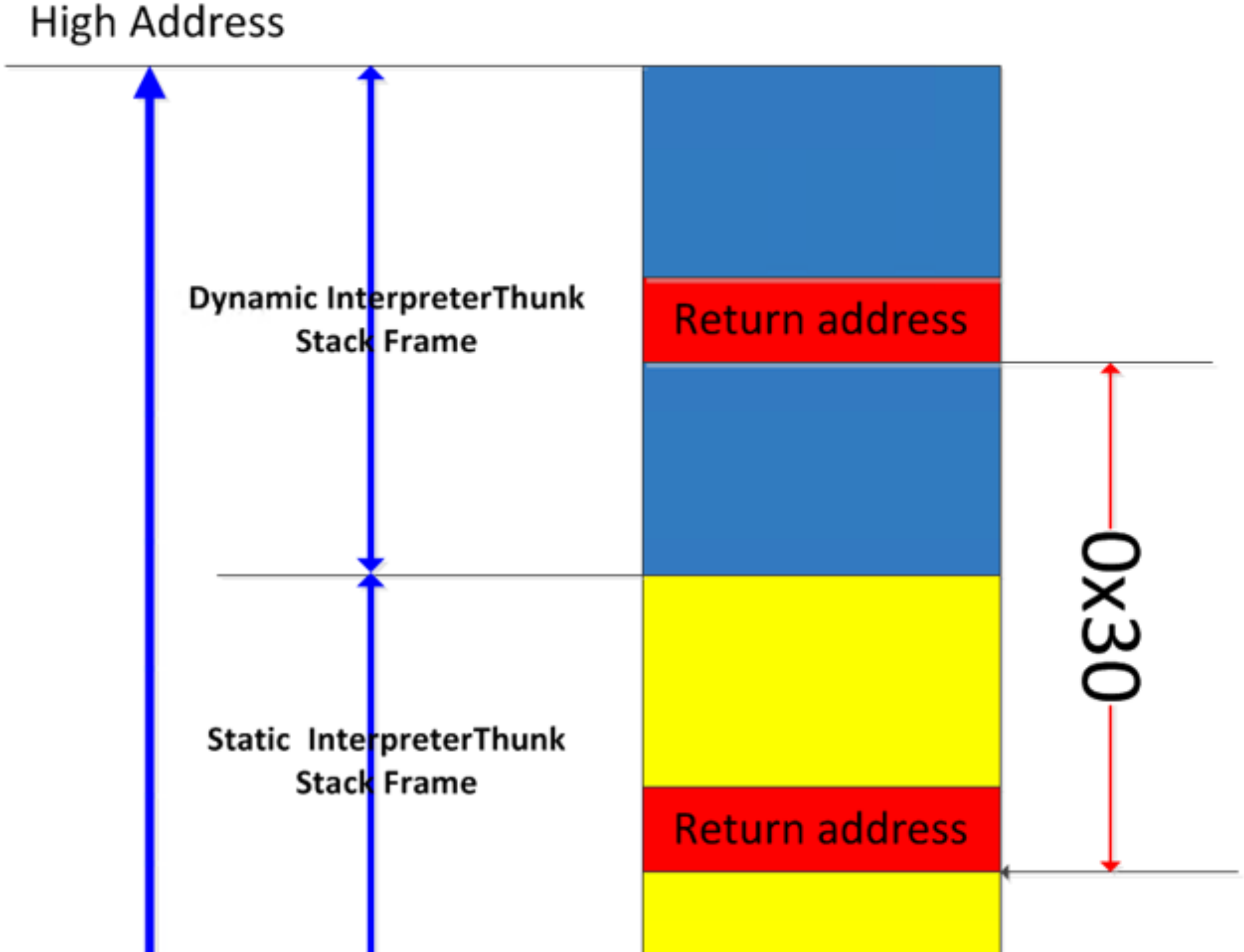
Exploit Vuln 4: Stackpivot function

Second:after xchg eax,esp run



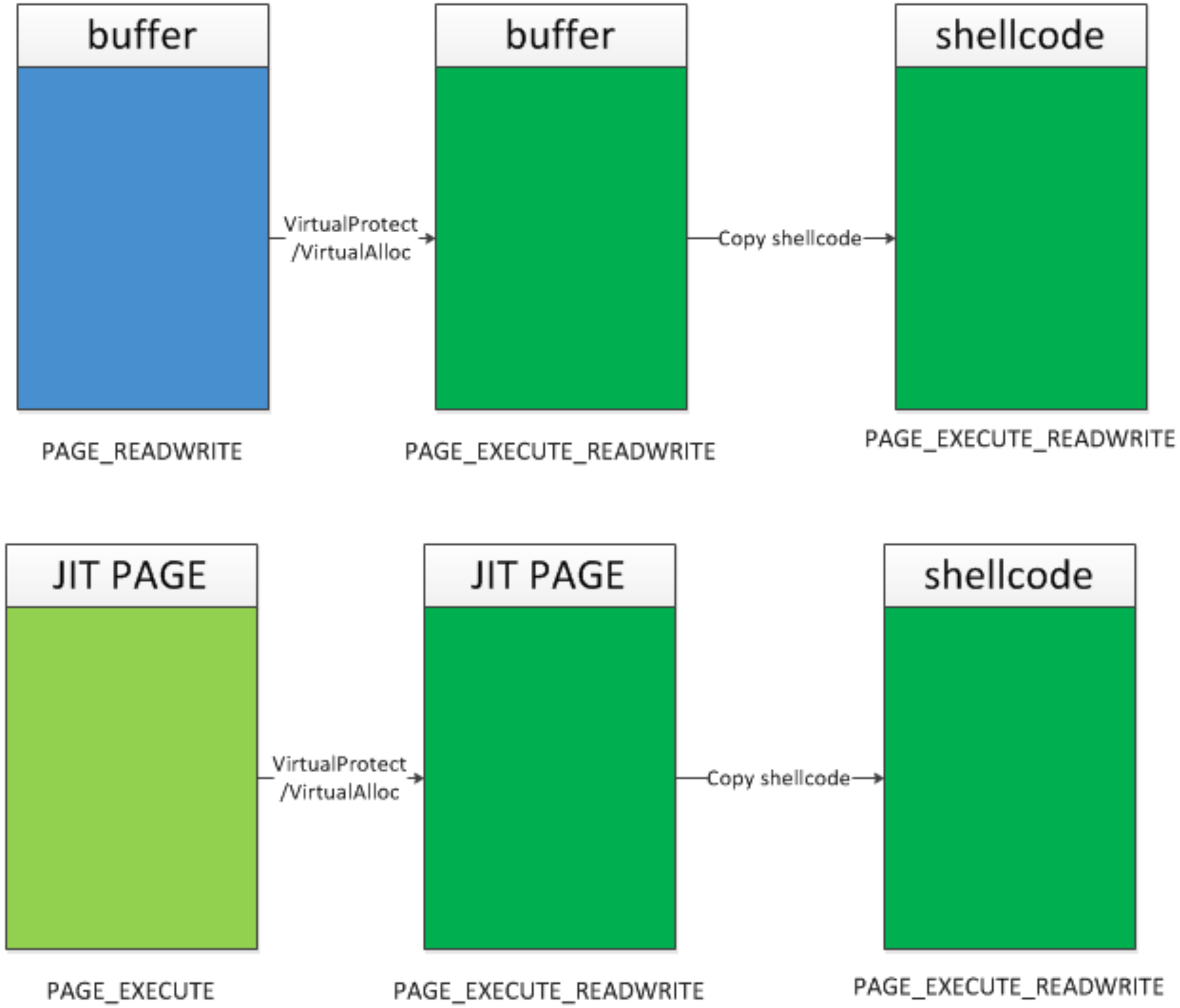
BYPASS RFG

- InterpreterStackFrame::InterpreterThunk
- eax, rax save the return value.


High Address
Dynamic InterpreterThunk Stack Frame
Return address
0x30
Static InterpreterThunk Stack Frame
Return address

InterpreterStackFrame
::addressofReturnAddress

VirtualAlloc/VirtualProtect Exploit



Improvements

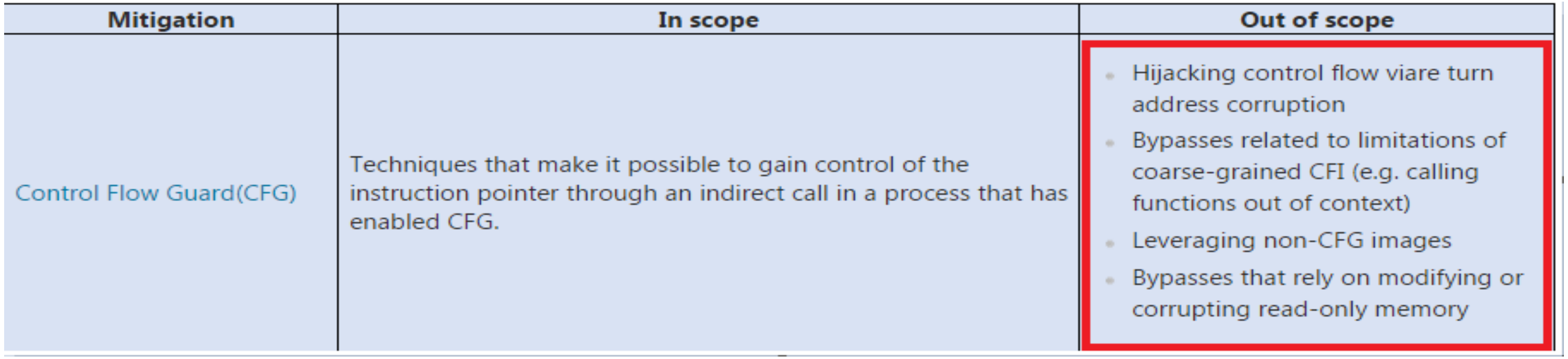
- Addressing CFG coverage gaps
- Disable RtlRemoteCall when CFG is enabled
- compiler directive: __declspec(guard(suppress))
- Setjmp/Longjmp hardening
- Arbitrary Code Guard

Arbitrary Code Guard

- **Not for CFG, actual effect on CFG have a great impact**
- **Prohibited to modified PAGE_EXECUTE to PAGE_EXECUTE_READWRITE**
- **Prohibited to modified PAGE_READWRITE to PAGE_EXECUTE_READWRITE**
- **Kill using Virtualalloc/VirtualProtect methods to bypass CFG.**

Exist Attack Surface

- Bypass that rely on modifying or corrupting read-only memory
  - _guard_check_icall_fptr
- write return address(RFG not enabled)
- CFG friendly API which is CFG valid
- Data Only Attack



Mitigation	In scope	Out of scope
Control Flow Guard(CFG)	Techniques that make it possible to gain control of the instruction pointer through an indirect call in a process that has enabled CFG.	Hijacking control flow viare turn address corruption Bypasses related to limitations of coarse-grained CFI (e.g. calling functions out of context) Leveraging non-CFG images Bypasses that rely on modifying or corrupting read-only memory

Acknowledgement

- **Jack Tang : Co-found MSRC 33966**
- **Kai Yu**

references

- Yunhai Zhang How To Avoid Implement An Exploit Friendly JIT
- David Weston、Matt Miller
  Windows 10 Mitigation Improvements
- Henry Li
  Control Flow Guard Improvements in Windows 10 Anniversary Update